

TMS320C4x

User's Guide

1996

Digital Signal Processing Solutions





SPRU063B



TMS320C4x User's Guide

SPRU063 March 1996







IMPORTANT NOTICE

Texas Instruments Incorporated (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Please be aware that TI products are not intended for use in life-support appliances, devices, or systems. Use of TI product in such applications requires the written approval of the appropriate TI officer. Certain applications using semiconductor devices may involve potential risks of personal injury, property damage, or loss of life. In order to minimize these risks, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. Inclusion of TI products in such applications is understood to be fully at the risk of the customer using TI devices or systems.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1996, Texas Instruments Incorporated

Preface

Read This First

About This Manual

This user's guide serves as a reference book for the TMS320C40 and TMS320C44 digital signal processors. Throughout the book, all references to the TMS320C4x apply to both devices, except when otherwise noted.

How to Use This Manual

The following table summarizes the information contained in this user's guide:

If you are looking for information about:	Turn to these chapters:
Addressing modes	Chapter 6, Addressing Modes
ARAUs	Chapter 2, Architectural Overview
Bootloader	Chapter 10, The Bootloader
Bus Structure	Chapter 2, Architectural Overview Chapter 9, External Bus Operation
Cache	Chapter 4, Memory and the Instruction Cache
Communication Ports	Chapter 12, Communication Ports
CPU Architecture	Chapter 2, Architectural Overview Chapter 3, CPU Registers
DMA	Chapter 11, The DMA Coprocessor
Data Formats	Chapter 5, Data Formats and Floating-Point Op- eration
Delayed Branches	Chapter 7, Program Flow Control
Instruction set	Chapter 14, Assembly Language Instructions

If you are looking for information about:	Turn to these chapters:
Interrupts	Chapter 7, Program Flow Control
Memory	Chapter 2, Architectural Overview
	Chapter 4, Memory and the Instruction Cache
Peripherals	Chapter 12, Communication Ports
	Chapter 11, The DMA Coprocessor
	Chapter 13, <i>Timers</i>
Overview of the 'C4x	Chapter 1, Introduction
Program control	Chapter 7, Program Flow Control
Pipeline	Chapter 8, Pipeline Operation
Registers	Chapter 3, CPU Registers
	Chapter 12, Communication Ports
	Chapter 11, The DMA Coprocessor
	Chapter 13, <i>Timers</i>
Repeat Mode	Chapter 7, Program Flow Control
Reset	Chapter 7, Program Flow Control
Timers	Chapter 13, <i>Timers</i>
Traps	Chapter 7, Program Flow Control

Style and Symbol Conventions

This document uses the following conventions:

Program listings, program examples, file names, and symbol names are shown in a special font. Examples use a bold version of the special font for emphasis. Here is a sample program listing segment:

*			
LOOP1	RPTB	MAX	
	CMPF	*AR0,R0	;Compare number to the maximum
MAX	LDFLT	*AR0,R0	; If greater, this is a new max
	В	NEXT	
LOOP2	RPTB	MIN	
	CMPF	*AR0++(1),R0	;Compare number to the minimum
MIN	LDFLT	*-AR0(1),R0	; If smaller, this is new minimum
NEXT	•		

□ In syntax descriptions, the instruction is in **bold face** and the parameters are in *italic face*. Portions of a syntax that are in **bold face** should be entered as shown; portions of a syntax that are in *italic face* describe the *type* of information that should be entered. Here is an example of an instruction:

CMPF3 src2, src1

Notice that although the instruction mnemonic (CMPF3 in this example) is in capital letters, the 'C4x assembler **is not case sensitive** — it can assemble mnemonics entered in either upper or lower case.

CMPF3 is the instruction mnemonic. This instruction has two parameters, indicated by *src2* and *src1*.

Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you must specify the information within the brackets; however, you don't enter the brackets themselves. Here's an example of an instruction that has an optional parameter:

[label] LDP src [,DP]

The **LDP** instruction is shown with two parameters; one is optional. The first parameter, *src*, is required. The second parameter, *DP*, and the label, are optional. As this syntax shows, if you use the optional second parameter, you must precede it with a comma.

Throughout this book MSB indicates the most significant bit and LSB indicates the least significant bit. MS indicates the most significant byte and LS indicates the least significant byte.

v

Information About Cautions and Warnings

This book may contain cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to <u>you</u>.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Related Documentation From Texas Instruments

The following books describe the TMS320 floating-point devices and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

- **TMS320C4x General-Purpose Applications User's Guide** (literature number SPRU159) describes software and hardware applications for the 'C4x processor. Also includes development support information, parts lists, and XDS510 emulator design considerations.
- **TMS320C4x Parallel Processing Development System Technical Reference** (literature number SPRU075) describes the TMS320C4x parallel processing system, a system with four C4xs with shared and distributed memory.
- **Parallel Processing with the TMS320C4x** (literature number SPRA031) describes parallel processing and how the 'C4x can be used in parallel processing. Also provides sample parallel processing applications.
- **TMS320 Floating-Point DSP Assembly Language Tools User's Guide** (literature number SPRU035) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C3x and 'C4x generations of devices.
- **TMS320 Floating-Point DSP Optimizing C Compiler User's Guide** (literature number SPRU034) describes the TMS320 floating-point C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C3x and 'C4x generations of devices.
- **TMS320C4x C Source Debugger User's Guide** (literature number SPRU054) tells you how to invoke the 'C4x emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.
- **TMS320C4x Technical Brief** (literature number SPRU076) gives a condensed overview of the 'C4x DSP and its development tools. It also lists TMS320C4x third parties.

- **TMS320 Family Development Support Reference Guide** (literature number SPRU011) describes the '320 family of digital signal processors and the various products that support it. This includes code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). This book also lists related documentation, outlines seminars and the university program, and gives factory repair and exchange information.
- **TMS320 Third-Party Support Reference Guide** (literature number SPRU052) alphabetically lists over 100 third parties that supply various products that serve the family of '320 digital signal processors—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.
- *TMS320 DSP Designer's Notebook: Volume 1 (SPRT125).* Presents solutions to common design problems using 'C2x, 'C3x, 'C4x, 'C5x, and other TI DSPs.

Related Articles and Books

A wide variety of related documentation is available on digital signal processing. These references fall into one of the following application categories:

- General-Purpose DSP
- Graphics/Imagery
- Speech/Voice
- Control
- Multimedia
- Military
- Telecommunications
- Automotive
- Consumer
- Medical
- Development Support

In the following list, references appear in alphabetical order according to author. The documents contain beneficial information regarding designs, operations, and applications for signal-processing systems; all of the documents provide additional references. Texas Instruments strongly suggests that you refer to these publications.

General-Purpose DSP:

1) Antoniou, A., *Digital Filters: Analysis and Design*, New York, NY: McGraw-Hill Company, Inc., 1979.

- 2) Brigham, E.O., *The Fast Fourier Transform*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1974.
- Burrus, C.S., and T.W. Parks, *DFT/FFT and Convolution Algorithms*, New York, NY: John Wiley and Sons, Inc., 1984.
- Chassaing, R., Horning, D.W., "Digital Signal Processing with Fixed and Floating-Point Processors." *CoED*, USA, Volume 1, Number 1, pages 1–4, March 1991.
- Defatta, David J., Joseph G. Lucas, and William S. Hodgkiss, *Digital Signal Processing: A System Design Approach*, New York: John Wiley, 1988.
- Erskine, C., and S. Magar, "Architecture and Applications of a Second-Generation Digital Signal Processor." *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, USA, 1985.
- Essig, D., C. Erskine, E. Caudel, and S. Magar, "A Second-Generation Digital Signal Processor." *IEEE Journal of Solid-State Circuits*, USA, Volume SC–21, Number 1, pages 86–91, February 1986.
- Frantz, G., K. Lin, J. Reimer, and J. Bradley, "The Texas Instruments TMS320C25 Digital Signal Microcomputer." *IEEE Microelectronics*, USA, Volume 6, Number 6, pages 10–28, December 1986.
- Gass, W., R. Tarrant, T. Richard, B. Pawate, M. Gammel, P. Rajasekaran, R. Wiggins, and C. Covington, "Multiple Digital Signal Processor Environment for Intelligent Signal Processing." *Proceedings of the IEEE,* USA, Volume 75, Number 9, pages 1246–1259, September 1987.
- Gold, Bernard, and C.M. Rader, *Digital Processing of Signals*, New York, NY: McGraw-Hill Company, Inc., 1969.
- Hamming, R.W., *Digital Filters*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.
- 12) IEEE ASSP DSP Committee (Editor), *Programs for Digital Signal Processing*, New York, NY: IEEE Press, 1979.
- 13) Jackson, Leland B., *Digital Filters and Signal Processing*, Hingham, MA: Kluwer Academic Publishers, 1986.
- Jones, D.L., and T.W. Parks, A Digital Signal Processing Laboratory Using the TMS32010, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.
- 15) Lim, Jae, and Alan V. Oppenheim, *Advanced Topics in Signal Processing*, Englewood Cliffs, NJ: Prentice- Hall, Inc., 1988.

- 16) Lin, K., G. Frantz, and R. Simar, Jr., "The TMS320 Family of Digital Signal Processors." *Proceedings of the IEEE*, USA, Volume 75, Number 9, pages 1143–1159, September 1987.
- 17) Lovrich, A., Reimer, J., "An Advanced Audio Signal Processor." Digest of Technical Papers for 1991 International Conference on Consumer Electronics, June 1991.
- 18) Magar, S., D. Essig, E. Caudel, S. Marshall and R. Peters, "An NMOS Digital Signal Processor with Multiprocessing Capability." *Digest of IEEE International Solid-State Circuits Conference*, USA, February 1985.
- 19) Morris, Robert L., *Digital Signal Processing Software*, Ottawa, Canada: Carleton University, 1983.
- 20) Oppenheim, Alan V. (Editor), *Applications of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.
- 21) Oppenheim, Alan V., and R.W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975 and 1988.
- Oppenheim, A.V., A.N. Willsky, and I.T. Young, *Signals and Systems*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.
- Papamichalis, P.E., and C.S. Burrus, "Conversion of Digit-Reversed to Bit-Reversed Order in FFT Algorithms." *Proceedings of ICASSP 89*, USA, pages 984–987, May 1989.
- Papamichalis, P., and R. Simar, Jr., "The TMS320C30 Floating-Point Digital Signal Processor." *IEEE Micro Magazine*, USA, pages 13–29, December 1988.
- 25) Parks, T.W., and C.S. Burrus, *Digital Filter Design*, New York, NY: John Wiley and Sons, Inc., 1987.
- 26) Peterson, C., Zervakis, M., Shehadeh, N., "Adaptive Filter Design and Implementation Using the TMS320C25 Microprocessor." *Computers in Education Journal*, USA, Volume 3, Number 3, pages 12–16, July–September 1993.
- Prado, J., and R. Alcantara, "A Fast Square-Rooting Algorithm Using a Digital Signal Processor." *Proceedings of IEEE*, USA, Volume 75, Number 2, pages 262–264, February 1987.
- 28) Rabiner, L.R. and B. Gold, *Theory and Applications of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.
- 29) Simar, Jr., R., and A. Davis, "The Application of High-Level Languages to Single-Chip Digital Signal Processors." *Proceedings of ICASSP 88*, USA, Volume D, page 1678, April 1988.

- 30) Simar, Jr., R., T. Leigh, P. Koeppen, J. Leach, J. Potts, and D. Blalock, "A 40 MFLOPS Digital Signal Processor: the First Supercomputer on a Chip." *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396–0, Volume 1, pages 535–538, April 1987.
- 31) Simar, Jr., R., and J. Reimer, "The TMS320C25: a 100 ns CMOS VLSI Digital Signal Processor." 1986 Workshop on Applications of Signal Processing to Audio and Acoustics, September 1986.
- 32) Texas Instruments, *Digital Signal Processing Applications with the TMS320 Family*, 1986; Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.
- 33) Treichler, J.R., C.R. Johnson, Jr., and M.G. Larimore, *A Practical Guide* to Adaptive Filter Design, New York, NY: John Wiley and Sons, Inc., 1987.

Graphics/Imagery:

- 1) Andrews, H.C., and B.R. Hunt, *Digital Image Restoration*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.
- Gonzales, Rafael C., and Paul Wintz, *Digital Image Processing*, Reading, MA: Addison-Wesley Publishing Company, Inc., 1977.
- Papamichalis, P.E., "FFT Implementation on the TMS320C30." *Proceedings of ICASSP 88*, USA, Volume D, page 1399, April 1988.
- Pratt, William K., Digital Image Processing, New York, NY: John Wiley and Sons, 1978.
- 5) Reimer, J., and A. Lovrich, "Graphics with the TMS32020." *WESCON/85 Conference Record*, USA, 1985.

Speech/Voice:

- DellaMorte, J., and P. Papamichalis, "Full-Duplex Real-Time Implementation of the FED-STD-1015 LPC-10e Standard V.52 on the TMS320C25." *Proceedings of SPEECH TECH 89*, pages 218–221, May 1989.
- Frantz, G.A., and K.S. Lin, "A Low-Cost Speech System Using the TMS320C17." *Proceedings of SPEECH TECH '87*, pages 25–29, April 1987.
- Gray, A.H., and J.D. Markel, *Linear Prediction of Speech*, New York, NY: Springer-Verlag, 1976.
- 4) Jayant, N.S., and Peter Noll, *Digital Coding of Waveforms*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.
- Papamichalis, Panos, Practical Approaches to Speech Coding, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.
- Papamichalis, P., and D. Lively, "Implementation of the DOD Standard LPC-10/52E on the TMS320C25." *Proceedings of SPEECH TECH '87*, pages 201–204, April 1987.

- Pawate, B.I., and G.R. Doddington, "Implementation of a Hidden Markov Model-Based Layered Grammar Recognizer." *Proceedings of ICASSP* 89, USA, pages 801–804, May 1989.
- Rabiner, L.R., and R.W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.
- Reimer, J.B. and K.S. Lin, "TMS320 Digital Signal Processors in Speech Applications." *Proceedings of SPEECH TECH '88*, April 1988.
- Reimer, J.B., M.L. McMahan, and W.W. Anderson, "Speech Recognition for a Low-Cost System Using a DSP." *Digest of Technical Papers for 1987 International Conference on Consumer Electronics*, June 1987.

Control:

- Ahmed, I., "16-Bit DSP Microcontroller Fits Motion Control System Application." *PCIM*, October 1988.
- Ahmed, I., "Implementation of Self Tuning Regulators with TMS320 Family of Digital Signal Processors." *MOTORCON '88*, pages 248–262, September 1988.
- Ahmed, I., and S. Lindquist, "Digital Signal Processors: Simplifying High-Performance Control." *Machine Design*, September 1987.
- Ahmed, I., and S. Meshkat, "Using DSPs in Control." Control Engineering, February 1988.
- Allen, C. and P. Pillay, "TMS320 Design for Vector and Current Control of AC Motor Drives." *Electronics Letters*, UK, Volume 28, Number 23, pages 2188–2190, November 1992.
- Bose, B.K., and P.M. Szczesny, "A Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System for Electric Vehicle Propulsion." *Proceedings of IECON '87*, Volume 1, pages 454–463, November 1987.
- Hanselman, H., "LQG-Control of a Highly Resonant Disc Drive Head Positioning Actuator." *IEEE Transactions on Industrial Electronics*, USA, Volume 35, Number 1, pages 100–104, February 1988.
- 8) Jacquot, R., *Modern Digital Control Systems*, New York, NY: Marcel Dekker, Inc., 1981.
- Katz, P., *Digital Control Using Microprocessors*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
- 10) Kuo, B.C., *Digital Control Systems*, New York, NY: Holt, Reinholt, and Winston, Inc., 1980.

- Lovrich, A., G. Troullinos, and R. Chirayil, "An All-Digital Automatic Gain Control." *Proceedings of ICASSP 88*, USA, Volume D, page 1734, April 1988.
- Matsui, N. and M. Shigyo, "Brushless DC Motor Control Without Position and Speed Sensors." *IEEE Transactions on Industry Applications*, USA, Volume 28, Number 1, Part 1, pages 120–127, January–February 1992.
- Meshkat, S., and I. Ahmed, "Using DSPs in AC Induction Motor Drives." Control Engineering, February 1988.
- 14) Panahi, I. and R. Restle, "DSPs Redefine Motion Control." *Motion Control Magazine*, December 1993.
- 15) Phillips, C., and H. Nagle, *Digital Control System Analysis and Design*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

Multimedia:

- 1) Reimer, J., "DSP-Based Multimedia Solutions Lead Way Enhancing Audio Compression Performance." *Dr. Dobbs Journal*, December 1993.
- Reimer, J., G. Benbassat, and W. Bonneau Jr., "Application Processors: Making PC Multimedia Happen." *Silicon Valley PC Design Conference*, July 1991.

Military:

 Papamichalis, P., and J. Reimer, "Implementation of the Data Encryption Standard Using the TMS32010." *Digital Signal Processing Applications*, 1986.

Telecommunications:

- Ahmed, I., and A. Lovrich, "Adaptive Line Enhancer Using the TMS320C25." *Conference Records of Northcon/86*, USA, 14/3/1–10, September/October 1986.
- Casale, S., R. Russo, and G. Bellina, "Optimal Architectural Solution Using DSP Processors for the Implementation of an ADPCM Transcoder." *Proceedings of GLOBECOM '89*, pages 1267–1273, November 1989.
- Cole, C., A. Haoui, and P. Winship, "A High-Performance Digital Voice Echo Canceller on a SINGLE TMS32020." *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243–4, Volume 1, pages 429–432, April 1986.
- 4) Cole, C., A. Haoui, and P. Winship, "A High-Performance Digital Voice Echo Canceller on a Single TMS32020." *Proceedings of IEEE*

International Conference on Acoustics, Speech and Signal Processing, USA, 1986.

- 5) Lovrich, A., and J. Reimer, "A Multi-Rate Transcoder." *Transactions on Consumer Electronics*, USA, November 1989.
- Lovrich, A. and J. Reimer, "A Multi-Rate Transcoder." Digest of Technical Papers for 1989 International Conference on Consumer Electronics, June 7–9, 1989.
- Lu, H., D. Hedberg, and B. Fraenkel, "Implementation of High-Speed Voiceband Data Modems Using the TMS320C25." *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396–0, Volume 4, pages 1915–1918, April 1987.
- Mock, P., "Add DTMF Generation and Decoding to DSP– μP Designs." *Electronic Design*, USA, Volume 30, Number 6, pages 205–213, March 1985.
- 9) Reimer, J., M. McMahan, and M. Arjmand, "ADPCM on a TMS320 DSP Chip." *Proceedings of SPEECH TECH 85*, pages 246–249, April 1985.
- Troullinos, G., and J. Bradley, "Split-Band Modem Implementation Using the TMS32010 Digital Signal Processor." *Conference Records of Electro/86 and Mini/Micro Northeast*, USA, 14/1/1–21, May 1986.

Automotive:

 Lin, K., "Trends of Digital Signal Processing in Automotive." *International Congress on Transportation Electronic (CONVERGENCE '88*), October 1988.

Consumer:

- 1) Frantz, G.A., J.B. Reimer, and R.A. Wotiz, "Julie, The Application of DSP to a Product." *Speech Tech Magazine*, USA, September 1988.
- Reimer, J.B., and G.A. Frantz, "Customization of a DSP Integrated Circuit for a Customer Product." *Transactions on Consumer Electronics*, USA, August 1988.
- Reimer, J.B., P.E. Nixon, E.B. Boles, and G.A. Frantz, "Audio Customization of a DSP IC." *Digest of Technical Papers for 1988 International Conference on Consumer Electronics*, June 8–10 1988.

Medical:

 Knapp and Townshend, "A Real-Time Digital Signal Processing System for an Auditory Prosthesis." *Proceedings of ICASSP 88*, USA, Volume A, page 2493, April 1988. 2) Morris, L.R., and P.B. Barszczewski, "Design and Evolution of a Pocket-Sized DSP Speech Processing System for a Cochlear Implant and Other Hearing Prosthesis Applications." *Proceedings of ICASSP 88*, USA, Volume A, page 2516, April 1988.

Development Support:

- 1) Mersereau, R., R. Schafer, T. Barnwell, and D. Smith, "A Digital Filter Design Package for PCs and TMS320." *MIDCON/84 Electronic Show and Convention*, USA, 1984.
- Simar, Jr., R., and A. Davis, "The Application of High-Level Languages to Single-Chip Digital Signal Processors." *Proceedings of ICASSP 88*, USA, Volume 3, pages 1678–1681, April 1988.

If You Need Assistance. . .

If you want to	Do this
Request more information about Texas Instruments Digital Signal Processing (DSP) products	Write to: Texas Instruments Incorporated Market Communications Manager MS 736 P.O. Box 1443 Houston, Texas 77251–1443
Order Texas Instruments documentation	Call the TI Literature Response Center: (800) 477–8924
Ask questions about product operation or report suspected problems	Contact the DSP hotline: Phone: (713) 274–2320 FAX: (713) 274–2324 Electronic Mail: 4389750@mcimail.com.
Obtain the source code in this user's guide.	Call the TI BBS: (713) 274–2323
	Ftp from: ftp.ti.com log in as user ftp cd to / mirrors/tms320bbs
Visit TI online, including TI&ME [™] , your own customized web page.	Point your browser at: http://www.ti.com
Report mistakes or make com- ments about this or any other TI	Send electronic mail to: comments@books.sc.ti.com
documentation.	Send printed comments to: Texas Instruments Incorporated Technical Publications Mgr., MS 702 P.O. Box 1443 Houston, Texas 77251–1443

Trademarks

MS is a registered trademark of Microsoft Corp. MS-Windows is a registered trademark of Microsoft Corp. MS-DOS is a registered trademark of Microsoft Corp. OS/2 is a trademark of International Business Machines Corp. Sun and SPARC are trademarks of Sun Microsystems, Inc. VAX and VMS are trademarks of Digital Equipment Corp.

Contents

4	lintra	J .	4 -	4
I	Introc	luces the	TMS320 family and the TMS320C4y	I
	1 1	TMCOO		0
	1.1	1111002	UC4X Devices	2
		1.1.1	The TMC320040	2
		1.1.2	The TMC0000 040	2
	10	1.1.3 Kay Fa	The TMS320L040	2
	1.2		alures of the TMS320C4X	3 4
	1.3	11/1532	0040 and TMS320044 Device Comparison 1-4	+
2	Archi	tectural	Overview	1
	Briefly	y describ	pes the architecture of the CPU, buses, interrupts, and peripherals of the 'C4x	
	2.1	Central	Processing Unit (CPU) 2	4
		2.1.1	Floating-Point/Integer Multiplier 2-4	4
		2.1.2	Arithmetic Logic Unit (ALU) and Internal Buses 2-4	4
		2.1.3	Auxiliary Register Arithmetic Units (ARAUs) 2-6	6
		2.1.4	CPU Primary Register File 2-6	6
		2.1.5	CPU Expansion Register File 2-10	0
	2.2	Memor	y Organization	1
		2.2.1	RAM, ROM, and Cache 2-1	1
		2.2.2	Memory Maps 2-13	3
		2.2.3	Memory Aliasing ('C44 only) 2-17	7
		2.2.4	Memory Addressing Modes 2-18	З
	2.3	Interna	Bus Operation	9
	2.4	Externa	al Bus Operation	0
	2.5	Interrup	ots	1
	2.6	Periphe	erals	2
		2.6.1	Communication Ports	3
		2.6.2	Direct Memory Access (DMA) Coprocessor 2-23	3
		2.6.3	Timers	4

Contents

3	CPU Lists regis	Registe and des ter file	rs cribes the contents of the CPU primary register file and the CPU expansion	. 3-1 n
	3.1	CPU P	rimarv Register File	. 3-2
	-	3.1.1	Extended-Precision Registers (R0–R11)	. 3-3
		3.1.2	Auxiliary Registers (AR0–AR7)	. 3-4
		3.1.3	Data-Page Pointer (DP)	. 3-4
		3.1.4	Index Registers (IR0, IR1)	. 3-4
		3.1.5	Block-Size Register (BK)	. 3-5
		3.1.6	System Stack Pointer (SP)	. 3-5
		3.1.7	Status Register (ST)	. 3-5
		3.1.8	DMA Coprocessor Interrupt Enable Register (DIE)	. 3-8
		3.1.9	CPU Internal Interrupt Enable Register (IIE)	3-11
		3.1.10	IIOF Flag Register (IIF)	3-13
		3.1.11	Block-Repeat (RS, RE) and Repeat-Count (RC) Registers	3-16
		3.1.12	Program Counter (PC)	3-16
		3.1.13	Reserved Bits and Compatibility	3-16
	3.2	CPU E	xpansion Register File	3-17
4	Mem	ory and	the Instruction Cache	. 4-1
	Desc	ribes the	structure of the memory map and the architecture of the instruction cache	
	4.1	Memor	у Мар	. 4-2
	4.2	Periphe	eral Bus Memory Map	4-5
		4.2.1	Local and Global Memory Interface Control Registers	4-6
		4.2.2	Analysis Module Registers	4-6
		4.2.3	Timer Registers	. 4-7
		4.2.4	Communication Port Memory Map	4-8
		4.2.5	DMA Coprocessor Registers	4-9
	4.3	Instruct	tion Cache	4-10
		4.3.1	Instruction Cache Architecture	4-10
		4.3.2	Cache Control Bits	4-12
		4.3.3	Using the Cache	4-13
		4.3.4	The LRU Cache Algorithm	4-14
5	Data	Formate	s and Floating-Point Operation	5-1
	Desc opera	ribes inte ations are	eger and floating-point data formats and discusses how some mathematica e performed on floating-point numbers	1/
	5.1	Signed	-Integer Formats	. 5-2
		5.1.1	Short Integer Format	. 5-2
		5.1.2	Single-Precision Integer Format	5-2
	5.2	Unsign	ed-Integer Formats	5-3
		5.2.1	Short Unsigned-Integer Format	5-3
		5.2.2	Single-Precision Unsigned-Integer Format	5-3

	5.3	Floating-Point Formats	. 5-4
		5.3.1 Short Floating-Point Format	. 5-5
		5.3.2 Single-Precision Floating-Point Format	. 5-6
		5.3.3 Extended-Precision Floating-Point Format	. 5-7
		5.3.4 Determining the Decimal Equivalent of a Floating-Point Number	. 5-8
		5.3.5 Conversion Between Floating-Point Formats	5-11
	5.4	Floating-Point Conversion (IEEE Std. 754)	5-13
		5.4.1 Converting IEEE Format to Twos-Complement 'C4x Floating-Point Format .	5-14
		5.4.2 Converting Twos-Complement 'C4x Floating-Point Format to IEEE Format .	5-17
	5.5	Floating-Point Multiplication	5-19
	5.6	Floating-Point Addition and Subtraction	5-23
	5.7	Normalization (NORM Instruction)	5-27
	5.8	Rounding (RND Instruction)	5-29
	5.9	Floating-Point-to-Integer Conversion (FIX Instruction)	5-31
	5.10	Integer-to-Floating-Point Conversion (FLOAT Instruction)	5-33
	5.11	Reciprocal (RCPF Instruction)	5-34
		5.11.1 Reciprocal Algorithm	5-35
	5.12	Reciprocal Square Root (RSQRF Instruction)	5-36
		Newton-Raphson Algorithm	5-37
6	Addre	essing Modes	. 6-1
	Desci	ribes addressing modes, using address registers, and stack managements in the 'C4'	x
	6.1	Addressing Types	. 6-2
	6.2	Register Addressing	. 6-3
	6.3	Direct Addressing	. 6-5
	6.4	Indirect Addressing	. 6-6
	6.5	Immediate Addressing	6-18
	6.6	PC-Relative Addressing	6-19
	6.7	Encoding of Addressing Modes	6-21
		6.7.1 General Addressing Modes	6-21
		6.7.2 Three-Operand Addressing Modes	6-22
		6.7.3 Parallel Addressing Modes	6-24
		6.7.4 Conditional-Branch Addressing Modes	6-25
	6.8	Circular Addressing	6-27
	6.9	Bit-Reversed Addressing	6-32
7	Prog	am Flow Control	. 7-1
	Desci	ribes software and hardware features that control how a program flows	
	7.1	Repeat Mode	. 7-2
		7.1.1 Control Bits	. 7-3
		7.1.2 Repeat-Mode Operation	. 7-3
		7.1.3 RPTB and RPTBD Instructions	. 7-4
		7.1.4 RPTS Instruction	. 7-5
		7.1.5 Repeat Mode Restriction Rules	. 7-6
		7.1.6 RC Register Value After Repeat Mode Completes	. 7-7
		7.1.7 Nesting Block Repeats	. 7-8

Contents xix

Contents

	7.2	Delaye	d Branches	7-9
		7.2.1	Delayed Branches Without Annulling	7-10
		7.2.2	Delayed Branches With Annulling	7-11
	7.3	Calls, 7	Traps, Branches, Jumps, and Returns	7-12
	7.4	Interrup	pts	7-15
		7.4.1	Interrupt Vector Table and Prioritization	7-15
		7.4.2	CPU Interrupt Control Bits	7-17
		7.4.3	Interrupt Processing	7-18
		7.4.4	CPU Interrupt Latency	7-20
		7.4.5	External Interrupts	7-21
	7.5	Traps		7-24
		7.5.1	Initialization of Traps and Interrupts	7-24
		7.5.2	Operation of Traps	7-24
		7.5.3	Overlapping the Trap and Interrupt Vector Tables	7-25
	7.6	DMA Ir	nterrupts	7-26
		7.6.1	DMA Interrupt Control Bits	7-26
		7.6.2	DMA Interrupt Processing	7-27
		7.6.3	CPU/DMA Interrupt Interaction	7-28
	7.7	Reset		7-29
		7.7.1	Reset's Effects on Pin States	7-29
		7.7.2	Reset Vector Location	7-35
		770	Additional Pacat Operations	7 05
		1.1.3		
8	Pinel	7.7.3	ration	7-30 8-1
8	Pipel	ine Ope	ration	7-35
8	Pipel Desci	ine Ope	ration	7-35 8-1
8	Pipel Desci 8.1	ribes and Pipelin	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure a Conflicte	7-35 8-1 8-2
8	Pipel <i>Desci</i> 8.1 8.2	ine Ope ribes and Pipeline Pipeline	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts	7-35 8-1 8-2 8-4
8	Pipel Desci 8.1 8.2	ribes and Pipelin 8.2.1	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts	7-35 8-1 8-2 8-4 8-4
8	Pipel Desci 8.1 8.2	ribes and Pipelin 8.2.1 8.2.2	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts	
8	Pipel <i>Desci</i> 8.1 8.2	r.7.3 ine Ope ribes and Pipelin 8.2.1 8.2.2 8.2.3 Momor	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts	
8	Pipel <i>Desci</i> 8.1 8.2 8.3	ribes and Pipelin Pipelin 8.2.1 8.2.2 8.2.3 Memor	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance	
8	Pipel <i>Desci</i> 8.1 8.2 8.3 8.4	ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Branch Conflicts	
8	Pipel <i>Desci</i> 8.1 8.2 8.3 8.4	ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data L cade and Storee	
8	Pipel Desci 8.1 8.2 8.3 8.4	r.7.3 ine Ope ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores	
8	Pipel Desc. 8.1 8.2 8.3 8.4 Exter	ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 mal Bus	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores	
8	Pipel Desci 8.1 8.2 8.3 8.4 Exter Desci	ribes and Pipelin Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 mal Bus ribes the	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores e Operation e features and functions of the two 'C4x external buses	
8	Pipel Desc. 8.1 8.2 8.3 8.4 Exter Desc. 9.1	ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 ribes the Overvie	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores e features and functions of the two 'C4x external buses	
8	Pipel Desc. 8.1 8.2 8.3 8.4 Exter Desc. 9.1 9.2	ribes and Pipelin Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 mal Bus ribes the Overvie Memor	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts Memory Conflicts ng of Memory Accesses Program Fetches Data Loads and Stores e Goperation e features and functions of the two 'C4x external buses ew y Interface Signals	
8	Pipel Desc. 8.1 8.2 8.3 8.4 Exter Desc. 9.1 9.2 9.3	ribes and Pipelin Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 ribes the Overvie Memor Memor	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores e features and functions of the two 'C4x external buses ew y Interface Signals y-Interface Control Registers	
9	Pipel Desc. 8.1 8.2 8.3 8.4 Exter Desc. 9.1 9.2 9.3	ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 rnal Bus ribes the Overvie Memor 9.3.1	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts y Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores e features and functions of the two 'C4x external buses ew y Interface Signals y-Interface Control Registers Mapping Addresses to Strobes	
9	Pipel Desc. 8.1 8.2 8.3 8.4 Exter Desc. 9.1 9.2 9.3	r.7.3 ine Ope ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 ribes the Overvie Memor 9.3.1 9.3.2	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Register Conflicts Memory Conflicts ry Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores e Goperation e features and functions of the two 'C4x external buses ew y Interface Signals y-Interface Control Registers Mapping Addresses to Strobes Page Size Operation	
9	Pipel Desci 8.1 8.2 8.3 8.4 Exter Desci 9.1 9.2 9.3 9.4	ribes and Pipelin 8.2.1 8.2.2 8.2.3 Memor Clockir 8.4.1 8.4.2 mal Bus ribes the Overvie Memor 9.3.1 9.3.2 Progra	ration d explains the operation of the four pipeline stages in the 'C4x CPU e Structure e Conflicts Branch Conflicts Branch Conflicts Memory Conflicts y Accesses for Maximum Performance ng of Memory Accesses Program Fetches Data Loads and Stores e features and functions of the two 'C4x external buses ew y Interface Signals y-Interface Control Registers Mapping Addresses to Strobes Page Size Operation mmable Wait States	

	9.6 9.7 9.8	Using Enable Signals to Control Signal Groups9-38Interlocked Operations9-399.7.1LDFI and LDII9.7.2STFI and STII9.7.3SIGI9.7.4Interlocked Examples9.7.5Bus-Lock Pins and Bus Timing9-409-419.7.59-429-449-45
10	The B	ootloader
	Descr	ibes 'C4x bootloader operation and also lists the bootloader code
	10.1	Bootloader Description
	10.2	Mode Selection
	10.3	Bootloading Sequence 10-5
	10.4	Bootloading from External Memory (Examples) 10-10
	10.5	Bootloading from a Communication Port (Examples) 10-16
	10.6	Modifying the IIOFx Pins After Bootloading 10-19
	10.7	The Bootloader Program
11	The D Descr	MA Coprocessor
	11.1	Introduction
	11.2	DMA Functional Description
	11.0	11.2.1 DMA Basic Operation
	11.3	DMA Registers 11-7 11.2.1 Control Pogistor 11.7
		11.3.2 Address and Index Begisters 11-15
		11.3.3 Transfer Counter and Auxiliary Transfer Counter Begisters 11-16
		11.3.4 Link Pointer and Auxiliary Link-Pointer Registers
	11.4	DMA Unified Mode 11-19
	11.5	DMA Split Mode 11-20
	11.6	DMA Internal Priority Schemes 11-22
		11.6.1 Fixed Priority Scheme 11-22
		11.6.2 Rotating Priority Scheme 11-22
		11.6.3 Split Mode and DMA Channel Arbitration 11-24
	11.7	CPU and DMA Coprocessor Arbitration
	11.8	Data Iranster Modes
		11.0.1 RUIHING IN TRANSFER MODE = 002
		11.8.3 Running in TRANSFER MODE = 102 (Autoinitialization 1) 11-29
		11.8.4 Running in TRANSFER MODE = 112 (Autoinitialization 7)

Contents

	11.9	Autoinitialization	11-34
		11.9.1 Unified Mode	11-35
		11.9.2 Split Mode	11-35
		11.9.3 Incrementing the Link Pointer	11-36
		11.9.4 Synchronization	11-37
		11.9.5 Effect on DMA Control Register Bits	11-38
		11.9.6 Consecutive Autoinitializations	11-40
	11.10	DMA and Interrupts	11-42
		11.10.1 Interrupts and Synchronization of DMA Channels	11-43
		11.10.2 Synchronization Mode Bits	11-46
	11.11	DMA Memory Transfer Timing	11-51
		11.11.1 Single DMA Memory Transfer Timing	11-51
		11.11.2 DMA Transfer Rate in Synchronization Mode	11-55
10	Com	numination Douto	10.1
12	Deser	inumcation Forts	12-1
	Descr		
	12.1	Features	. 12-2
	12.2		. 12-3
		12.2.1 Token Transfer Operation	. 12-5
	10.0	12.2.2 Data Transfer Operation	. 12-6
	12.3	Memory Map and Registers	. 12-7
		12.3.1 Communication-Port Control Register (CPCR)	. 12-8
			. 12-9
		12.3.3 Output-Port Register	. 12-9
	10.1	12.3.4 Communication-Port Software Reset Register	12-10
	12.4	Port Arbitration Units (PAUs)	12-11
	12.5		12-14
		12.5.1 Input FIFO Hall Operation	12-15
	10.6	Coordinating Communication Parts With the CPU and DMA Converses	12-13
	12.0	Token Transfer Operation	12-17
	12.7		12-19
	12.0	CSTRB Width Restrictions	12-22
	12.9	Synchronizers	12-26
	12.10	Module Beset	12-29
	12.11	Tips for Using Communication Ports	12-32
_			
13	Timer	S	. 13-1
	Descr	ibes and discusses operation of the two 'C4x on-chip timers	
	13.1	Overview of the Timers	. 13-2
	13.2	Timer Pins	. 13-4

	13.3	Timer (Control Registers	13-5
		13.3.1	Timer Control Register	13-6
		13.3.2	Timer Period Register	13-7
		13.3.3	Timer Counter Register	13-8
		13.3.4	Boundary Conditions in the Control Registers	13-8
	13.4	Timer F	Pulse Generation	13-9
	13.5	Timer I	nterrupts	13-11
		13.5.1	Timer Interrupts and Their Vectors	13-11
		13.5.2	Timer Interrupt Operation	13-11
		13.5.3	Considerations When Using a Timer Interrupt	13-12
	13.6	Selecti	ng CLKSRC and FUNC Values	13-13
		13.6.1	CLKSRC = 1 and FUNC = 0	13-13
		13.6.2	CLKSRC=1 and FUNC=1	13-13
		13.6.3	CLKSRC = 0 and FUNC = 0	13-14
		13.6.4	CLKSRC = 0 and FUNC = 1	13-14
	13.7	Using T	FCLKx as General-Purpose I/O Pins	13-15
	13.8	Configu	uring a Timer	13-16
14	Asse	mbly La	nguage Instructions	14-1
14	Asse Lists t	mbly La	nguage Instructions	14-1
14	Asser Lists t 14.1	mbly La the entire Instruct	nguage Instructions e instruction set for the 'C4x tion Set	14-1
14	Asser <i>Lists t</i> 14.1	mbly La the entire Instruct 14.1.1	e instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions	14-1 14-2 14-2
14	Assei Lists t 14.1	mbly La the entire Instruct 14.1.1 14.1.2	e instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions	14-1 14-2 14-2 14-4
14	Asser Lists t 14.1	mbly La the entire Instruct 14.1.1 14.1.2 14.1.3	e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions	14-1 14-2 14-2 14-4 14-4
14	Asser Lists t 14.1	mbly La the entire Instruct 14.1.1 14.1.2 14.1.3 14.1.3	e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions	14-1 14-2 14-2 14-4 14-6 14-7
14	Asser Lists t 14.1	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.3 14.1.4 14.1.5	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions	14-1 14-2 14-2 14-4 14-6 14-7 14-8
14	Asser Lists t 14.1	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.3 14.1.4 14.1.5 14.1.6	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions	14-1 14-2 14-2 14-4 14-6 14-7 14-8 14-9
14	Asser Lists t 14.1	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.3 14.1.4 14.1.5 14.1.6 14.1.7	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions Illegal Instructions	14-1 14-2 14-2 14-4 14-6 14-7 14-8 14-9 14-11
14	Asse <i>Lists</i> 14.1	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.3 14.1.4 14.1.5 14.1.6 14.1.7 Conditi	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions Illegal Instructions on Codes and Flags	14-2 14-2 14-4 14-6 14-7 14-8 14-9 14-11
14	Asse <i>Lists</i> 1 14.1 14.2 14.2 14.3	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.4 14.1.5 14.1.6 14.1.7 Conditi Individu	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions Illegal Instructions on Codes and Flags ual Instruction Descriptions	14-2 14-2 14-4 14-6 14-7 14-8 14-9 14-11 14-12 14-16
14	Asser <i>Lists i</i> 14.1 14.2 14.2	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.4 14.1.5 14.1.6 14.1.7 Conditi Individu 14.3.1	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions Illegal Instructions on Codes and Flags ual Instruction Descriptions Symbols and Abbreviations	14-2 14-2 14-4 14-6 14-7 14-8 14-9 14-11 14-12 14-16 14-16
14	Asser <i>Lists t</i> 14.1 14.2 14.2	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.4 14.1.5 14.1.6 14.1.7 Conditi Individu 14.3.1 14.3.2	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions Illegal Instructions on Codes and Flags Jual Instruction Descriptions Optional Assembler Syntaxes	14-1 14-2 14-2 14-4 14-6 14-7 14-8 14-9 14-11 14-12 14-16 14-16 14-18
14	Asset <i>Lists i</i> 14.1 14.2 14.3	mbly La the entire 14.1.1 14.1.2 14.1.3 14.1.4 14.1.5 14.1.6 14.1.7 Conditi Individu 14.3.1 14.3.2 14.3.3	Inguage Instructions e instruction set for the 'C4x tion Set Load-and-Store Instructions Two-Operand Instructions Three-Operand Instructions Program Control Instructions Interlocked Operations Instructions Parallel Operations Instructions Illegal Instructions on Codes and Flags ual Instruction Descriptions Symbols and Abbreviations Optional Assembler Syntaxes Individual Instruction Descriptions	14-1 14-2 14-4 14-6 14-7 14-8 14-9 14-11 14-12 14-16 14-16 14-18 14-20

Figures

2–1	TMS320C4x Block Diagram
2–2	Central Processing Unit (CPU) 2-5
2–3	Memory Organization
2–4	'C40 Memory Map 2-14
2–5	C44 Memory Map
2–6	Peripheral Memory Map 2-16
2–7	Memory Aliasing (C44 only) 2-17
2–8	Peripheral Modules
3–1	Extended-Precision Register Floating-Point Format
3–2	Extended-Precision Register Integer Format
3–3	Status Register (ST)
3–4	DMA Interrupt Enable Register Bit Functions for DMA Unified Mode
3–5	DMA Interrupt Enable Register Bit Functions for DMA Split Mode
3–6	Internal Interrupt Enable Register (IIE)
3–7	Interrupt Flag Register (IIF)
4–1	'C40 Memory Map 4-3
4–2	'C44 Memory Map 4-4
4–3	Peripheral Memory Map 4-5
4–4	Memory Interface Control Registers 4-6
4–5	Timer Registers
4–6	Communication Port Memory Map 4-8
4–7	DMA Coprocessor Memory Map 4-9
4–8	Address Partitioning for Cache Control Algorithm 4-10
4–9	Instruction Cache Architecture 4-11
5–1	Short-Integer Format and Sign Extension of Short Integer 5-2
5–2	Single-Precision Integer Format 5-2
5–3	Short Unsigned-Integer Format and Zero Fill 5-3
5–4	Single-Precision Unsigned-Integer Format 5-3
5–5	General Floating-Point Format 5-4
5–6	Short Floating-Point Format 5-5
5–7	Single-Precision Floating-Point Format 5-6
5–8	Extended-Precision Floating-Point Format 5-7
5–9	Short Floating-Point Format Conversion to Single-Precision Floating-Point Format 5-11
5–10	Short Floating-Point Format Conversion to Extended-Precision Floating-Point Format . 5-11
5–11	Single-Precision Floating-Point Format Conversion to Extended-Precision Floating-Point Format

5–12	Extended-Precision Floating-Point Format Conversion to Single-Precision Floating-Point Format	5-12
5–13	IEEE Single-Precision Std. 754 Floating-Point Format	5-13
5–14	'C4x Single-Precision Twos-Complement Floating-Point Format	5-13
5–15	Flowchart for Floating-Point Multiplication	5-20
5–16	Flowchart for Floating-Point Addition	5-24
5–17	Flowchart for NORM Instruction Operation	5-27
5-18	Flowchart for Floating-Point Bounding by the BND Instruction	5-30
5-19	Flowchart for Floating-Point-to-Integer Conversion by FIX Instruction	5-32
5-20	Flowchart for Integer-to-Floating-Point Conversion by FLOAT Instructions	5-33
5_21	BCPE Instruction Algorithm	5-34
5_22	BSOBE Instruction Algorithm	5-37
6_1		6-5
6_2	Indiroct Addrossing Operand Encoding	. 0-5 6-6
6.2	Encoding for 24 Bit PC Polotivo Addrossing Mode	6 20
0-3 6 4	Encoding for Conorol Addressing Modes	6 20
0-4 6 5	Encoding for Type 1 Three Operand Addressing Modes (C2y and C4y)	6 24
C-0	Encoding for Type 1 Three-Operand Addressing Modes (C3x and C4x)	0-24
ю—ю с 7	Encoding for Type 2 Three-Operand Addressing Modes (C4X Only)	0-24
6-7	Encoding for Parallel Multiply With ADD/SUB	6-24
6-8	Encoding for Conditional-Branch Addressing Modes	6-26
6-9	Register Relationships in Circular Addressing	6-28
6–10		6-29
6–11		6-30
6–12	Data Structure for FIR Filters	6-31
7–1	CALL Response Timing	7-14
7–2	Interrupt-Vector Table (IVT)	7-16
7–3	IIF Register Modification	7-18
7–4	CPU Interrupt Processing	7-19
7–5	Flow of Traps	7-24
7–6	Trap Vector Table (TVT)	7-25
7–7	DMA Interrupt Processing	7-27
7–8	Parallel CPU and DMA Interrupt Processing	7-28
8–1	Pipeline Structure	. 8-3
8–2	Two-Operand Instruction Word	8-20
8–3	Three-Operand Instruction Word	8-20
8–4	Multiply or CPU Operation With a Parallel Store	8-21
8–5	Two Parallel Stores	8-22
8–6	Parallel Multiplies and Adds	8-23
9–1	Global and Local Memory Interface Control Signals	. 9-3
9–2	Location of the Memory-Interface Control Registers	. 9-7
9–3	Fields in the Memory-Interface Control Registers	. 9-7
9–4	Effects of STRB ACTIVE on Global Memory Bus Memory Map	9-12
9–5	STRBx PAGESIZE Fields Example	9-13
9–6	STRB and RDY Timing	9-16

Figures

9–7	Read Same Page, Read Same Page, Write Same Page Sequence	9-18
9–8	Write Same Page, Write Same Page, Read Same Page Sequence	9-19
9–9	Read Same Page, Read Different Page, Read Same Page Sequence	9-20
9–10	Write Same Page, Write Different Page, Write Same Page Sequence	9-21
9–11	Write Same Page, Read Different Page, Write Different Page Sequence	9-22
9–12	Read Different Page, Read Different Page, Write Same Page Sequence	9-23
9–13	Write Different Page, Write Different Page, Read Same Page Sequence	9-24
9–14	Read Same Page, Write Different Page, Read Different Page Sequence	9-25
9–15	Read Same Page, Idle One Cycle, Read Same Page Sequence	9-26
9–16	Write Same Page, Idle One Cycle, Write Different Page Sequence	9-27
9–17	Idle, Read Different Page, Idle Sequence	9-28
9–18	Idle, Write Same Page, Idle Sequence	9-29
9–19	Write Different or Same Page, Idle, Idle Sequence	9-30
9–20	Read Same Page on STRB1, STRB0, and on STRB1 Sequence When STRB SWITCH = 0	9-31
9–21	Read Same Page on STRB1, STRB0, Read Different Page on STRB1 Sequence When STRB SWITCH = 0	9-32
9–22	Read Same Page on STRB1, STRB0, and on STRB1 Sequence When STRB SWITCH = 1	9-33
9–23	Read Same Page on STRB1, STRB0, Read Different Page on STRB1 Sequence When STRB SWITCH = 1	9-34
9–24	Write Same Page on STRB1, STRB0, Read Same Page on STRB1	9-35
9–25	Read With One Wait State	9-36
9–26	Write With One Wait State	9-37
9–27	Using Enable Signals to Put Signal Groups in a High-Impedance State	9-38
9–28	Multiple 'C4x Devices Sharing Global Memory	9-42
9–29	LDII or LDFI External Access	9-45
9–30	LDII or LDFI and STII or STFI External Access	9-46
9–31	SIGI External Access Timing	9-47
9–32	SIGI When LOCK Is Already Low	9-48
9–33	IACK Timing	9-50
10–1	Mode Selection Flow	10-4
10–2	Memory Load Flow	10-6
10–3	Communication-Port Load Mode Flow	10-7
10–4	Circuit for Generation of a Low IIOF Signal for Bootloader Selection	10-19
11–1	DMA Coprocessor Memory Map	11-4
11–2	DMA Channel Control Register	11-8
11–3	DMA Coprocessor Address Generation	11-16
11–4	Transfer Counter Registers	11-17
11–5	Link Pointer Registers	11-18
11–6	Typical Unified-Mode DMA Channel Configuration	11-19
11–7	Typical Split-Mode DMA Configuration	11-21
11–8	Rotating Priority Mode Example of the DMA Coprocessor	11-23
11–9	Rotating Priority DMA Read and Write Sequence Example (Unified Mode)	11-23

11-1	0 Example of a Priority Wheel	11-24
11-1	1 Example of a Channel Priority Scheme in Split Mode	11-25
11-1	2 Service Sequence for Split Mode Priority Example	11-26
11-1	3 DMA Channel Running in Transfer Mode 102 (Autoinitialization Method 1a)	11-29
11–1	4 DMA Channel Running in Transfer Mode 102 (Autoinitialization Method 1b)	11-30
11–1	5 DMA Channel Running in Transfer Mode 112 (Autoinitialization Method 2a)	11-31
11–1	6 DMA Channel Running in Transfer Mode 112 (Autoinitialization Method 2b)	11-33
11–1	7 Store New Values of DMA Channel Registers in Memory (SPLIT MODE = 0)	11-35
11–1	8 Store New Values of DMA Channel Registers in Memory (SPLIT MODE = 1 and Transfer Counter = 0)	11-36
11–1	9 Store New Values of DMA Channel Registers in Memory (SPLIT MODE = 1 and Auxiliary Transfer Counter = 0)	11-36
11–2	20 DMA Channel Control Register Bits Modifiable by Autoinitialization in Unified Mode	11-39
11–2	21 DMA Channel Control Register Bit Modifiable by Autoinitialization of the Primary Channel in Split Mode	11-40
11-2	2 DMA Channel Control Register Bits That Can Be Modified by	
	Autoinitialization of the Auxiliary Channel in Split Mode	11-40
11–2	23 Self-Referential Link Pointer	11-41
11–2	24 Referring to a New Link Pointer	11-41
11–2	25 DIE Register Bit Functions for DMA Unified Mode	11-44
11–2	26 DIE Register Bit Functions for DMA Split Mode	11-45
11–2	7 No DMA Synchronization	11-47
11–2	28 DMA Source Synchronization	11-48
11–2	29 DMA Destination Synchronization	11-49
11–3	0 Unified Mode DMA Source and Destination Synchronization	11-50
11–3	1 Timing and Number of Cycles for DMA Transfers to On-Chip Destination	11-52
11–3	2 Timing and Number of Cycles for DMA Transfers to a Local-Bus Destination	11-53
11–3	3 Timing and Number of Cycles for DMA Transfers to a Global-Bus Destination	11-54
11–3	4 Unified-Mode DMA Timing for Different Synchronizations	11-55
11–3	5 Split-Mode DMA Timing for Different Synchronizations	11-56
12-	Communication Port Block Diagram	. 12-4
12–2	2 'C4x Communication-Port Interface-Connection Example	. 12-5
12–3	B Communication-Port Memory Map	. 12-7
12–4	Communication-Port Control Register (CPCR)	. 12-8
12–	5 Communication-Port Arbitration-Unit State Diagram	12-12
12–6	5 Token Transfer Operation	12-20
12–7	Word Transfer Operation	12-23
12-8	3 Type-One Synchronizer Minimum Delay	12-26
12–9	Type-One Synchronizer Maximum Delay	12-26
12-	0 Type-Two Synchronizer Minimum Delay	12-27
12-	1 Type-Two Synchronizer Maximum Delay	12-27
12-1	2 Type-Three Synchronizer Minimum Delay	12-27
12-1	3 Type-Three Synchronizer Maximum Delay	12-28
12-	4 Post-Reset State for an Output Port	12-30

Contents xxvii

Figures

12–15	Post-Reset State for an Input Port	12-31
13–1	Timer Block Diagram	. 13-3
13–2	Memory-Mapped Timer Locations	. 13-5
13–3	Timer Control Register	. 13-6
13–4	Timer Pulse Mode and Clock Mode Timing	. 13-9
13–5	Timer Output Generation Examples	13-10
13–6	Timer Configuration With CLKSRC=1 and FUNC=0	13-13
13–7	Timer Configuration With CLKSRC = 1 and FUNC = 1	13-13
13–8	Timer Configuration With CLKSRC = 0 and FUNC = 0	13-14
13–9	Timer Configuration With CLKSRC = 0 and FUNC = 1	13-14
13–10	TCLK as an Input (I/O = 0)	13-15
13–11	TCLK as an Output (I/O = 1)	13-15
14–1	Status Register	14-13

Tables

	Comparison of 2010 and 2011 Eastures	1 /
1-1 0 1	COmparison of 040 and 044 realities	1-4
2-1	CPU Primary Register Filo	2-7
3-3	Summary of the CE and CE Bite	3-2
3_3	DMA Chappels 0 and 1 (DMA0 and DMA1) Unified Mode Synchronization Interrupts	3-7
0-0 2 1	DMA Channels 2 to 5 (DMA2 to DMA5) Unified Mode Synchronization Interrupts	3-3
0-4 2 5	DMA Channels 2 to 5 (DMA2 to DMA5) Onlined Mode Synchronization Interrupts	2 10
3-0 2 6	DMA Channels 0 and 1 (DMA0 and DMA1) Split-Mode Synchronization Interrupts	2 11
0-0 27	CPU Expansion Registere	. 3-11
3-7 4 1	Combined Effect of the CE and CE Dite	. 3-17
4-1 5 1	Converting IEEE Formet to Twee Complement Floating Deint Formet	. 4-13
5-1 5-0	Converting Titles Complement Electing Deint Format to IEEE Format	. 5-14
0-2 c 1	Conventing Twos-Complement Floating-Point Format to IEEE Format	. 5-17
0-1 C 0	Ladireet Addressing	6-3
6 2	Three Operand Instruction Addressing Medee	0-7
6-3	Infee-Operand Instruction Addressing Modes	. 6-22
6-4 7 1	Index Steps and Bit-Reversed Addressing	. 6-33
7-1		7-2
7-2	Dia Otatao At Oustan Depat	. 7-21
7-3		. 7-29
/-4		. 7-35
8–1	One Program Fetch and One Data Access for Maximum Performance	. 8-1/
8–2	One Program Fetch and Two Data Accesses for Maximum Performance	. 8-18
9–1		9-4
9–2	Global Memory Port Status for STRB0 and STRB1 Accesses	9-5
9–3	Page Size as Defined by STRB0/1 PAGESIZE Bits	9-9
9–4	Address Ranges Specified by STRB ACTIVE Bits	. 9-10
9–5	Address Ranges Specified by LSTRB ACTIVE Bits	. 9-11
9–6	Wait-State Generation for Each Value of SWW	. 9-15
9–7	Interlocked Operations	. 9-39
10–1	Bootloader Mode Selection Using Pins IIOF(3–0)	. 10-3
10–2	Structure of Source Program Data Stream	. 10-8
10–3	Byte-Wide Configured Memory	10-11
10–4	16-Bit Wide Configured Memory	10-14
10–5	32-Bit Wide Configured Memory	10-15
11–1	DMA PRI Bits and CPU/DMA Arbitration Rules	11-12
11–2	TRANSFER MODE (AUX TRANSFER MODE) Field Descriptions	11-12

Contents xxix

Tables

11–3	SYNC MODE Field Descriptions in Unified Mode	11-13
11–4	SYNC MODE Field Descriptions in Split Mode	11-13
11–5	START (AUX START) Field Descriptions	11-14
11–6	STATUS (AUX STATUS) Field Descriptions	11-14
11–7	DMA PRI Bits and CPU/DMA Arbitration Rules	11-27
11–8	TRANSFER MODE (AUX TRANSFER MODE) Field Descriptions	11-28
11–9	Effect of SYNC MODE and AUTOINIT MODE Bits in Autoinitialization	11-38
11–10	DMA Channels 0 and 1 (DMA0 and DMA1) Unified-Mode	
	Synchronization Interrupts	11-44
11–11	DMA Channels 2 to 5 (DMA2 to DMA5) Unified-Mode Synchronization Interrupts	11-45
11–12	DMA Channels 0 and 1 (DMA0 and DMA1) Split-Mode Synchronization Interrupts	11-46
11–13	DMA Channels 2 to 5 (DMA2 to DMA5) Split-Mode Synchronization Interrupts	11-46
12–1	Communication-Port Software Reset Address ('C44 and 'C40 \geq 5.0)	12-10
12–2	PAU State Definitions	12-11
12–3	Summary of Input and Output FIFO Halting	12-14
12–4	Token Transfer Sequence	12-21
12–5	Word Transfer Sequence	12-24
12–6	Communication-Port Signals and Synchronizer Delays	12-28
14–1	Load-and-Store Instructions	. 14-3
14–2	Two-Operand Instructions	. 14-4
14–3	Three-Operand Instructions	. 14-6
14–4	Program Control Instructions	. 14-7
14–5	Interlocked Operations Instructions	. 14-8
14–6	Parallel Instructions	. 14-9
14–7	Output Value Formats	14-12
14–8	Condition Codes and Flags	14-14
14–9	Instruction Symbols	14-17
14–10	CPU Register Symbols	14-21

Examples

4–1	Enabling the Cache	4-13
5–1	Positive Number	. 5-9
5–2	Negative Number	5-10
5–3	Fractional Number	5-10
5–4	IEEE to 'C4x Conversion Within Block Memory Transfer	5-16
5–5	'C4x to IEEE Conversion Within Block Memory Transfer	5-18
5–6	Floating-Point Multiply (Both Mantissas = -2.0)	5-21
5–7	Floating-Point Multiply (Both Mantissas = 1.5)	5-21
5–8	Floating-Point Multiply (Both Mantissas = 1.0)	5-22
5–9	Floating-Point Multiply Between Positive and Negative Numbers	5-22
5–10	Floating-Point Addition	5-25
5–11	Floating-Point Subtraction	5-25
5–12	Floating-Point Addition With a 32-Bit Shift	5-26
5–13	Floating-Point Addition/Subtraction and Zero	5-26
5–14	NORM Instruction	5-28
5–15	Newton-Raphson Algorithm for Computing the Reciprocal	5-35
5–16	Newton-Raphson Algorithm for Computing the Reciprocal Square Root	5-38
6–1	Direct Addressing	. 6-5
6–2	Auxiliary Register Indirect	. 6-9
6–3	Indirect With Predisplacement Add	6-9
6–4	Indirect With Predisplacement Subtract	6-10
6–5	Indirect With Predisplacement Add and Modify	6-10
6–6	Indirect With Predisplacement Subtract and Modify	6-11
6–7	Indirect With Postdisplacement Add and Modify	6-11
6–8	Indirect With Postdisplacement Subtract and Modify	6-12
6–9	Indirect With Postdisplacement Add and Circular Modify	6-12
6–10	Indirect With Postdisplacement Subtract and Circular Modify	6-13
6–11	Indirect With Preindex Add	6-13
6–12	Indirect With Preindex Subtract	6-14
6–13	Indirect With Preindex Add and Modify	6-14
6–14	Indirect With Preindex Subtract and Modify	6-15
6–15	Indirect With Postindex Add and Modify	6-15
6–16	Indirect With Postindex Subtract and Modify	6-16
6–17	Indirect With Postindex Add and Circular Modify	6-16
6–18	Indirect With Postindex Subtract and Circular Modify	6-17
6–19	Indirect With Postindex Add and Bit-Reversed Modify	6-17
-	,	

Contents xxxi

Examples

6–20	Immediate Addressing	. 6-18
6–21	PC-Relative Addressing	. 6-19
6–22	FIR Filter Code Using Circular Addressing	. 6-31
6–23	Bit-Reversed Addressing Example	. 6-32
7–1	Repeat-Mode Control Algorithm	7-4
7–2	RPTB Operation	7-4
7–3	Incorrectly Placed Standard Branch	7-6
7–4	Incorrectly Placed Delayed Branch	7-7
7–5	Pipeline Conflict in a RPTB Instruction	7-7
7–6	Incorrectly Placed Delayed Branches	. 7-10
7–7	Delayed Branch Execution	. 7-10
8–1	Standard Branch	8-5
8–2	Delayed Branch Without Annul Option	8-6
8–3	Using BcondAF and BcondAT Instructions	8-7
8–4	Write to an AR Followed by an AR for Address Generation	8-8
8–5	A Read of ARs Followed by ARs for Address Generation	8-9
8–6	Program Wait Until CPU Data Access Completes	. 8-11
8–7	Program Wait Due to Multicycle Access	. 8-12
8–8	Multicycle Program Memory Fetches	. 8-12
8–9	Single Store Followed by Two Reads	. 8-13
8–10	Parallel Store Followed by Single Read	. 8-14
8–11	Busy External Port	. 8-15
8–12	Multicycle Data Reads	. 8-16
8–13	Conditional Calls and Traps	. 8-16
9–1	Busy-Waiting Loop	. 9-42
9–2	Task Counter Manipulation	. 9-42
9–3	Implementation of V(S)	. 9-43
9–4	Implementation of P(S)	. 9-43
10–1	Booting a 'C4x Multiprocessor System	10-17
12–2	Communication Port Reset	12-10
13–1	Maximum Frequency Timer Clock Setup	13-16

Chapter 1

Introduction

The TMS320C4x devices are 32-bit floating-point digital signal processors optimized for parallel processing. The 'C4x family combines a high performance CPU and DMA controller with up to six communication ports to meet the needs of multiprocessor and I/O-intensive applications. All 'C4x devices are compatible with TI's multi-chip development environment. Each device contains an on-chip analysis module, which supports hardware breakpoints for parallelprocessing development and debugging. The 'C4x family is source-code compatible with the TMS320C3x family of floating-point DSPs.

Торі	c Page
1.1	TMS320C4x Devices 1-2
1.2	Key Features of the TMS320C4x 1-3
1.3	TMS320C40 and TMS320C44 Device Comparison 1-4
1.1 TMS320C4x Devices

The TMS320C4x family is made up of three different members: the TMS320C40, the TMS320LC40, and the TMS320C44.

1.1.1 The TMS320C40

The TMS320C40 is the original member of the 'C4x family. It features a CPU that can deliver up to 30 MIPS/60 MFLOPS with a maximum I/O bandwidth of 384M bytes/s. The 'C40 has 2K words of on-chip RAM, 128 words of program cache and a bootloader. Two external buses provide an address reach of 4 gi-gawords of unified memory space. The 'C40 is available in a 325-pin CPGA package.

1.1.2 The TMS320C44

The TMS320C44 is a lower cost version of the 'C40, for parallel processing applications that are more price sensitive. The 'C44 features four communication ports and has an external address reach of 32M words over two external buses. To further reduce cost, the 'C44 comes in a 304-pin PQFP package. The TMS320C44 can deliver up to 30 MIPS/60 MFLOPS performance with a maximum I/O bandwidth of 384M bytes/s. The 'C44 is source-code compatible with the 'C40.

1.1.3 The TMS320LC40

The TMS320LC40 is the newest member of the 'C4x family. It is a low-power version of the 'C40 capable of delivering up to 40 MIPS/80 MFLOPS with a maximum I/O bandwidth of 488M bytes/s for high performance multiprocessing applications. The 'LC40 is source-code compatible with the 'C40 and 'C44.

Note:

See the chapter, *Development Support and Part Order Information*, in the *TMS320C4x General-Purpose Applications User's Guide* for device speeds, device availability information and part numbers.

1.2 Key Features of the TMS320C4x

The TMS320C4x has several key features:

- Up to 40 MIPS/80 MFLOPS performance with 488-Mbytes/s I/O capability
 - IEEE floating-point conversion for ease of use
 - Register-based CPU
 - Single-cycle byte and half-word manipulation capabilities
 - Divide and square root support for improved performance
- On-chip memory includes 2K words of SRAM, 128 words of program cache, and bootloader

Two external buses providing an address reach of up to 4 gigawords

Two memory-mapped 32-bit timers

- 6 and 12 channel DMA
- Up to six communication ports for multiprocessor communication
- Idle mode for reduced power consumption

1.3 TMS320C40 and TMS320C44 Device Comparison

Table 1–1 shows the major differences in features of the 'C40 and 'C44.

Table 1–1. Comparison of 'C40 and 'C44 Features

Feature	'C40	'C44
External local address bus	31 pins	24 pins
External global address bus	31 pins	24 pins
Address reach	4G × 32	$32M \times 32$
Number of comm ports	6	4
Commport direction pin	no	yes
NMI with bus grant feature	yes (for revisions \geq 5.0)	yes
Individual comm port reset	yes (for revisions \geq 5.0)	yes
Package	325-pin CPGA	304-pin PQFP

Chapter 2

Architectural Overview

The 'C4x's high performance is achieved through the precision and wide dynamic range of the floating-point units, on-chip memory, a high degree of parallelism, communication ports, and the DMA coprocessor.

This chapter gives an architectural overview of the 'C4x processor. Figure 2–1 is a block diagram of the 'C4x.

Торі	c Page)
2.1	Central Processing Unit (CPU) 2-4	
2.2	Memory Organization 2-11	
2.3	Internal Bus Operation 2-19	
2.4	External Bus Operation 2-20	
2.5	Interrupts 2-21	
2.6	Peripherals	



Figure 2–1. TMS320C4x Block Diagram



Figure 2–1. TMS320C4x Block Diagram (Continued)

Architectural Overview 2-3

2.1 Central Processing Unit (CPU)

The 'C4x's CPU has a register-based architecture. The CPU consists of the several components:

- Floating-point/integer multiplier
- Arithmetic Logic Unit (ALU)
- 32-bit barrel shifter
- □ Internal buses (CPU1/CPU2 and REG1/REG2)
- Auxiliary register arithmetic units (ARAUs)
- CPU register file

Figure 2–2 shows the CPU's components.

2.1.1 Floating-Point/Integer Multiplier

The multiplier performs single-cycle multiplications on 32-bit integer and 40-bit floating-point values. The 'C4x implementation of floating-point arithmetic allows for floating-point operations at fixed-point speeds via a 25-ns instruction cycle and a high degree of parallelism. To gain even higher throughput, you can use parallel instructions to perform a multiply and ALU operation in a single cycle.

When the multiplier performs floating-point multiplication, the inputs are 40-bit floating-point numbers, and the result is a 40-bit floating-point number. When the multiplier performs integer multiplication, the input data is 32 bits and yields either the 32 most-significant bits or the 32 least-significant bits of the resulting 64-bit product. See Chapter 5, *Data Formats and Floating-Point Operation*, for detailed information on data formats and floating-point operation.

2.1.2 Arithmetic Logic Unit (ALU) and Internal Buses

The ALU performs single-cycle operations on 32-bit integer, 32-bit logical, and 40-bit floating-point data, including single-cycle integer and floating-point conversions. Results of the ALU are always maintained in 32-bit integer or 40-bit floating-point formats. The barrel shifter is used to shift up to 32 bits left or right in a single cycle.

Four internal buses, CPU1, CPU2, REG1, and REG2, carry two operands from memory and two operands from the register file, thus allowing parallel multiplies and adds/subtracts on four integer or floating-point operands in a single cycle.

Figure 2–2. Central Processing Unit (CPU)



Architectural Overview 2-5

2.1.3 Auxiliary Register Arithmetic Units (ARAUs)

The two auxiliary register arithmetic units (ARAU0 and ARAU1) can generate two addresses in a single cycle. The ARAUs operate in parallel with the multiplier and ALU. They support addressing with displacements, index registers (IR0 and IR1), and circular and bit-reversed addressing. See Chapter 6, *Addressing Modes*, for a description of addressing modes.

2.1.4 CPU Primary Register File

The 'C4x primary register file provides 32 registers in a multiport register file that is tightly coupled to the CPU. Table 2–1 lists register names and functions, followed by the section number and page of each description.

All of the primary register file registers can be operated upon by the multiplier and ALU and can be used as general-purpose registers. However, the registers also have some special functions. For example, the 12 extended-precision registers are especially suited for maintaining floating-point results. The eight auxiliary registers support a variety of indirect addressing modes and can be used as general-purpose 32-bit integer and logical registers. The remaining registers provide system functions such as addressing, stack management, processor status, interrupts, and block repeat. See Chapter 3, *CPU Registers*, for detailed information about CPU registers. See Chapter 6, *Addressing Modes*, for information about register usage in addressing.

The **extended-precision registers (R0–R11)** are capable of storing and supporting operations on 32-bit integer and 40-bit floating-point numbers. Any instruction that assumes that the operands are floating-point numbers uses bits 39–0. If the operands are either signed or unsigned integers, only bits 31–0 are used, and bits 39–32 remain unchanged. This is true for all shift operations. See Chapter 5, *Data Formats and Floating-Point Operation*, for extended-precision register formats of floating-point and integer numbers.

The 32-bit **auxiliary registers (AR0–AR7)** can be accessed by the CPU and modified by the two auxiliary register arithmetic units (ARAUs). The primary function of the auxiliary registers is the generation of 32-bit addresses. They can also be used as loop counters or as 32-bit general-purpose registers that can be modified by the multiplier and ALU. See Chapter 6, *Addressing Modes*, for detailed information and examples of the use of auxiliary registers in addressing.

Assembler Syntax	Assigned Function Name	Subsection	Page
R0	Extended-precision register 0	3.1.1	3-3
R1	Extended-precision register 1	3.1.1	3-3
R2	Extended-precision register 2	3.1.1	3-3
R3	Extended-precision register 3	3.1.1	3-3
R4	Extended-precision register 4	3.1.1	3-3
R5	Extended-precision register 5	3.1.1	3-3
R6	Extended-precision register 6	3.1.1	3-3
R7	Extended-precision register 7	3.1.1	3-3
R8	Extended-precision register 8	3.1.1	3-3
R9	Extended-precision register 9	3.1.1	3-3
R10	Extended-precision register 10	3.1.1	3-3
R11	Extended-precision register 11	3.1.1	3-3
AR0	Auxiliary register 0	3.1.2	3-4
AR1	Auxiliary register 1	3.1.2	3-4
AR2	Auxiliary register 2	3.1.2	3-4
AR3	Auxiliary register 3	3.1.2	3-4
AR4	Auxiliary register 4	3.1.2	3-4
AR5	Auxiliary register 5	3.1.2	3-4
AR6	Auxiliary register 6	3.1.2	3-4
AR7	Auxiliary register 7	3.1.2	3-4
DP	Data-page pointer	3.1.3	3-4
IR0	Index register 0	3.1.4	3-4
IR1	Index register 1	3.1.4	3-4
BK	Block-size register	3.1.5	3-5
SP	System stack pointer	3.1.6	3-5

Table 2–1. CPU Primary Registers

Architectural Overview 2-7

Assembler Syntax	Assigned Function Name	Subsection	Page
ST	Status register	3.1.7	3-5
DIE	DMA Coprocessor interrupt enable	3.1.8	3-8
IIE	Internal-interrupt enable register	3.1.9	3-11
lIF	IIOF flag register	3.1.10	3-13
RS	Repeat start address	3.1.11	3-16
RE	Repeat end address	3.1.11	3-16
RC	Repeat counter	3.1.11	3-16

Table 2–1. CPU Primary Registers (Continued)

The **data page pointer (DP**) is a 32-bit register. The 16 LSBs of the data page pointer are used by the direct addressing mode as a pointer to the page of data being addressed. The 'C4x can address up to 64K pages, each page containing 64K words. Use of the data page pointer is described in subsection 6.3, *Direct Addressing*, on page 6-5.

The 32-bit **index registers** contain the value used by the auxiliary register arithmetic unit (ARAU) to compute an indexed address. See Section 6.4, *Indirect Addressing*, on page 6-6, and Section 6.9, *Bit-Reversed Addressing*, on page 6-32, for more information about the ARAU.

The ARAU uses the 32-bit **block size register (BK)** in circular addressing to specify the data block size. Circular addressing is described in Section 6.8, *Circular Addressing*, on page 6-27.

The **system stack pointer (SP)** is a 32-bit register that contains the address of the top of the system stack. The SP always points to the last element pushed onto the stack. A push performs a preincrement, and a pop performs a post-decrement of the system stack pointer. The SP is manipulated by interrupts, traps, calls, returns, and the PUSH/PUSHF and POP/POPF instructions. See Section 1.4, *System and User Stack Management*, in the *TMS320C4x General-Purpose Applications User's Guide* for information about managing the stacks.

The **status register (ST)** contains global information related to the state of the CPU. Typically, operations set the condition flags of the status register according to whether the result is zero, negative, etc. This includes register load and store operations as well as arithmetic and logical functions. When the status register is loaded, however, a bit-for-bit replacement is performed with the contents of the source operand, regardless of the state of any bits in the source operand. Therefore, following a load, the contents of the status register are identically equal to the contents of the source operand. This allows the status register to be easily saved and restored. See subsection 3.1.7, *Status Register (ST)*, on page 3-5, for definitions of the status register bits.

The **DMA coprocessor interrupt enable register (DIE)** is a 32-bit register containing 2- and 3-bit fields to designate the interrupt synchronization scheme for each of the six DMA channels. It allows each DMA channel to service a corresponding input communication port and output communication port. Also, each DMA channel can be synchronized with external interrupts or the on-chip timers. This register is described in subsection 3.1.8, *DMA Coprocessor Interrupt Enable Register (DIE)*, on page 3-8.

The **CPU internal interrupt enable register (IIE)** is a 32-bit register that enables/disables interrupts for the six communication ports, both timers, and the six DMA coprocessor channels. The IIE is described in subsection 3.1.9, *CPU Interrupt Enable Register (IIE)*, on page 3-11.

The **IIOF flag register (IIF)** controls the function (general-purpose I/O or interrupt) of the four external pins (IIOF0 to IIOF3). It also contains timer/DMA interrupt flags. Subsection 3.1.10, *IIOF Flag Register (IIF)*, on page 3-13, provides further description of this register.

The 32-bit **repeat counter (RC)** register specifies the number of times a block of code is to be repeated when a block repeat is performed. When the processor is operating in the repeat mode, the 32-bit **repeat start address register (RS)** contains the starting address of the block of program memory to be repeated, and the 32-bit **repeat end address register (RE)** contains the ending address of the block to be repeated. Further information about these registers is in subsection 3.1.11, *Block Repear (RS,RE) and Repeat Count (RC) Registers*, on page 3-16.

The **program counter (PC)** is a 32-bit register containing the address of the next instruction to be fetched. Although the PC is not part of the CPU register file, it is a register that can be modified by instructions that modify the program flow.

2.1.5 CPU Expansion Register File

Besides the CPU primary register file, the expansion register file contains two special registers that act as pointers:

- □ The IVTP register points to the interrupt-vector table (IVT), which defines vectors for all interrupts.
- □ The TVTP register points to the trap vector table (TVT), which defines vectors for 512 traps.

These two registers are fully described in Section 3.2, *CPU Expansion Register File* on page 3-17.

2.2 Memory Organization

The total memory reach of the 'C4x is 4G 32-bit words. Program memory (onchip RAM or ROM and external memory) as well as registers affecting timers, communication ports, and DMA channels are contained within this space. This allows tables, coefficients, program code, and data to be stored in either RAM or ROM. Thus, memory usage is maximized, and memory space allocated as desired.

By manipulating one external pin (ROMEN), you can configure the first onemegaword area of memory (0000 0000h to 000F FFFFh) to address the local address bus or to address the on-chip ROM when you use the bootloader (with remaining space reserved). This capability is further discussed in Section 4.1, *Memory Map*, on page 4-2.

2.2.1 RAM, ROM, and Cache

Figure 2–3 shows how the memory is organized on the 'C4x. RAM blocks 0 and 1 are 4K bytes ($1K \times 32$ bits) each. The ROM block is reserved and contains a bootloader. Each RAM and ROM block is capable of supporting two accesses in a single cycle. The separate program buses, data buses, and DMA buses allow for parallel program fetches, data reads and writes, and DMA operations. For example: the CPU can access two data values in one RAM block and perform an external program fetch in parallel with the DMA coprocessor loading another RAM block, all within a single cycle.

The reserved ROM block (upper right in Figure 2–3) contains a bootloader. This loader supports loading of program and data at reset time. Loading is from 8-, 16-, or 32-bit wide memories or any one of the six communication ports. Chapter 10, *The Bootloader*, explains the bootloader in detail.

A 128 \times 32-bit instruction cache is provided to store often-repeated sections of code, thus greatly reducing the number of needed off-chip accesses. This allows for code to be stored off-chip in slower, lower-cost memories. By using the cache to execute your program, the external buses are freed for use by the DMA controller or CPU.

For further information about memory and the instruction cache, see Section 4.1, *Memory Organization*, and Section 4.3, *Cache Memory*.



Figure 2–3. Memory Organization

2.2.2 Memory Maps

The memory map for each processor is shown in Figure 2–4 ('C40) and Figure 2–5 ('C44); for each processor, the level at the external pin ROMEN determines whether or not the first megaword of memory addresses the internal ROM or external memory. The maps illustrate the entire address space of the 'C40 and 'C44.

The value of ROMEN affects only the first megaword of memory:

- □ A 1 at external pin ROMEN causes internal ROM to be enabled at 0000h with the one-megaword space reserved (0000 0000h 000F FFFFh). This is shown in the right side of the figure.
- □ A 0 at ROMEN causes addresses 0000 0000h 000F FFFFh to be accessible on the local bus. This is shown in the left side of the figure.

The rest of the memory map is the same for either level of ROMEN:

- ☐ The second megaword of memory is devoted to peripherals (as shown in Figure 2–6).
- ☐ The third megaword of memory contains the two 1K-word (4K-byte) blocks of RAM (BLK0 and BLK1 as shown at 002F F800h − 002F FFFFh).
- □ The rest of the first 2 gigawords (0030 0000h 7FFF FFFFh) is on the local bus (external).
- The second 2 gigawords (8000 0000h FFFF FFFFh) are on the global bus (external).

Section 4.1, *Memory Map*, on page 4-2 describes the memory maps in greater detail. Section 9.2, *Memory Interface Signals* on page 9-3, and Section 9.3, *Memory Interface Control Registers* on page 9-6, discuss the local and global interfaces to memory. The peripheral bus map and the vector locations for reset, interrupts, and traps are also explained in those sections.

Caution

Any access to a reserved area in the address space produces unpredictable results. Do not attempt to access reserved areas.



Figure 2–4. 'C40 Memory Map

Figure 2–5. 'C44 Memory Map



Architectural Overview 2-15

Figure 2–6. Peripheral Memory Map

Address	Peripheral	Described in
0010 0000h 0010 000Fh	Local and global port control (16 words)	Subsection 4.2.1, Figure 4–4, page 4-6
0010 0010h 0010 001Fh	Analysis block registers (16 words)	Subsection 4.2.2.
0010 0020h 0010 002Fh	Timer 0 registers (16 words)	Subsection 4.2.3, Figure 4–5, page
0010 0030h 0010 003Fh	Timer 1 registers (16 words)	4-7
0010 0040h 0010 004Fh	Communication port 0 (16 words) ('C40 only)	Subsection 4.2.4, Figure 4–6, page
0010 0050h 0010 005Fh	Communication port 1 (16 words)	4-8
0010 0060h 0010 006Fh	Communication port 2 (16 words)	
0010 0070h 0010 007Fh	Communication port 3 (16 words) ('C40 only)	
0010 0080h 0010 008Fh	Communication port 4 (16 words)	
0010 0090h 0010 009Fh	Communication port 5 (16 words)	
0010 00A0h 0010 00AFh	DMA coprocessor channel 0 (16 words)	Subsection 4.2.5, Figure 4–7, page
0010 00B0h 0010 00BFh	DMA coprocessor channel 1 (16 words)	4-9
0010 00C0h 0010 00CFh	DMA coprocessor channel 2 (16 words)	
0010 00D0h 0010 00DFh	DMA coprocessor channel 3 (16 words)	
0010 00E0h 0010 00EFh	DMA coprocessor channel 4 (16 words)	
0010 00F0h 0010 00FFh	DMA coprocessor channel 5 (16 words)	

2.2.3 Memory Aliasing ('C44 only)

Memory aliasing occurs in the 'C44, since both the global and local ports on that device have 24 pins, instead of the 31 pins on each port in the 'C40. Memory aliasing causes the first 16 M of each address space to be repeated in the memory map. Memory on the local bus occupies, and is aliased, in the first 2 G of address space, and memory on the global bus occupies, and is aliased, in the second 2 G of address space. Figure 2–7 shows the alias regions on the local and global buses.

Figure 2–7. Memory Aliasing ('C44 only)



2.2.4 Memory Addressing Modes

The 'C4x supports a base set of general-purpose instructions as well as arithmetic-intensive instructions that are particularly suited for digital signal processing and other numeric-intensive applications. Refer to Chapter 6, *Addressing Modes*, for detailed information on addressing.

Four groups of addressing modes are provided on the 'C4x. Each group uses two or more of several different addressing types. The following list shows the addressing modes with their addressing types.

- General addressing modes:
 - Register. The operand is a CPU register.
 - Immediate. The operand is a 16-bit immediate value.
 - Direct. The operand is the contents of a 32-bit address (concatenation of 16 bits of the data page pointer and a 16-bit operand).
 - Indirect. A 32-bit auxiliary register indicates the address of the operand.
- Three-operand addressing modes:
 - **Register.** (same as for general addressing mode).
 - Indirect. (same as for general addressing mode).
 - **Immediate.** The operand is an 8-bit immediate value.
- Parallel addressing modes:
 - **Register.** The operand is an extended-precision register.
 - Indirect. (same as for general addressing mode).
- Branch addressing modes:
 - Register. (same as for general addressing mode).
 - PC-relative. A signed 16-bit displacement or a 24-bit displacement is added to the PC.

2.3 Internal Bus Operation

A large portion of the 'C4x's high performance is due to internal busing and parallelism. Separate buses allow for parallel program fetches, data accesses, and DMA accesses:

- Program buses PADDR and PDATA
- Data buses DADDR1, DADDR2, and DDATA
- DMA buses DMAADDR and DMADATA

These buses connect all of the physical spaces (on-chip memory, off-chip memory, and on-chip peripherals) supported by the 'C4x. Figure 2–3 shows these internal buses and their connections to on-chip and off-chip memory blocks.

The program counter (PC) is connected to the 32-bit program address bus (PADDR). The instruction register (IR) is connected to the 32-bit program data bus (PDATA). In this configuration, the buses can fetch a single instruction word every machine cycle.

The 32-bit data address buses (DADDR1 and DADDR2) and the 32-bit data data bus (DDATA) support two data memory accesses every machine cycle. The DDATA bus carries data to the CPU over the CPU1 and CPU2 buses. The CPU1 and CPU2 buses can carry two data memory operands to the multiplier, ALU, and register file every machine cycle. Also internal to the CPU are register buses REG1 and REG2, which can carry two data values from the register file to the multiplier and ALU every machine cycle. Figure 2–2 shows the buses that are internal to the CPU section of the processor.

The DMA controller is supported with a 32-bit address bus (DMAADDR) and a 32-bit data bus (DMADATA). These buses allow the DMA to perform memory accesses in parallel with the memory accesses occurring from the data and program buses.

2.4 External Bus Operation

The 'C4x provides two identical external interfaces: the global memory interface and the local memory interface. Each consists of a 32-bit data bus, a 31-bit ('C40) or 24-bit ('C44) address bus, and two sets of control signals. Both buses can be used to address external program/data memory or I/O space. The buses also have external RDY signals for wait-state generation with wait states inserted under software control. Chapter 9, *External Bus Operation*, covers external bus operation.

For multiple processors to access global memory and share data in a coherent manner, arbitration is necessary. This arbitration (handshaking) is the purpose of the 'C4x's interlocked operations, handled through **interlocked instructions**. For more information about interlocked instructions, see Section 9.7 on page 9-39, *Interlocked Operations*.

2.5 Interrupts

The 'C4x supports four external interrupts ($\overline{IIOF3}$ –0), a number of internal interrupts, a nonmaskable external \overline{NMI} interrupt, and a nonmaskable external \overline{RESET} signal, which sets the processor to a known state. The DMA and communication ports have their own internal interrupts. When the CPU responds to the interrupt, the IACK pin can be used to signal an external interrupt acknowledge. Section 7.4, on page 7-15, *Interrupts*, covers RESET and interrupt processing.

2.6 Peripherals

All 'C4x on-chip peripherals are controlled through memory-mapped registers on a dedicated peripheral bus. This peripheral bus is composed of a 32-bit data bus and a 32-bit address bus. This peripheral bus permits straightforward communication to the peripherals. The 'C4x peripherals include two timers and six ('C40) or four ('C44) communication ports. Figure 2–8 shows the peripherals with associated buses and signals.

Figure 2–8. Peripheral Modules



2.6.1 Communication Ports

Six ('C40) or four ('C44) high-speed communication ports provide rapid processor-to-processor communication through each port's dedicated communication interfaces. Coupled with the 'C4x's two memory interfaces (global and local), this allows you to construct a parallel processor system that attains optimum system performance by distributing tasks among several processors. Each 'C4x can pass the results of its work to another 'C4x through a communication port, enabling each 'C4x to continue working. Chapter 12, *Communication Ports*, explains communication port operation in detail.

The communication ports offer several features:

- 160-megabits/s (20-Mbytes or 5-Mwords per second) bidirectional data transfer operations (at 40-ns cycle time)
- Simple processor-to-processor communication via eight data lines and four control lines
- Buffering of all data transfers, both input and output
- Automatic arbitration to ensure communication synchronization
- Synchronization between the CPU or the direct-memory access (DMA) coprocessor and the six communication ports via internal interrupts and internal ready signals.
- Port direction pin (CDIR) to ease interfacing ('C44 only)

2.6.2 Direct Memory Access (DMA) Coprocessor

The six channels of the on-chip DMA coprocessor can read from or write to any location in the memory map without interfering with the operation of the CPU. This allows interfacing to slow external memories and peripherals without reducing throughput to the CPU. The DMA coprocessor contains its own address generators, source and destination registers, and transfer counter. Dedicated DMA address and data buses allow for minimization of conflicts between the CPU and the DMA coprocessor. A DMA operation consists of a block or single-word transfer to or from memory. A key feature of the DMA coprocessor is its ability to automatically reinitialize each channel following a data transfer. See Chapter 11, *The DMA Coprocessor*, for detailed information on the DMA coprocessor.

2.6.3 Timers

The two timer modules are general-purpose 32-bit timer/event counters with two signaling modes and internal or external clocking. They can signal internally to the 'C4x or externally to the outside world at specified intervals, or they can count external events. Each timer has an I/O pin that can be used as an input clock to the timer, as an output signal driven by the timer, or as a general-purpose I/O pin. The timers are described in detail in Chapter 13, *The Timers*.

Chapter 3

CPU Registers

The CPU *primary register file* contains 32 registers that can be used as operands by the multiplier and ALU (arithmetic logic unit). The register file includes the auxiliary registers, extended-precision registers, and index registers. These registers support addressing, floating-point/integer operations, stack management, processor status, block repeats, branching, and interrupts.

The CPU *expansion register file* contains two registers — the interrupt vector table pointer (IVTP) and the trap vector table pointer (TVTP).

This chapter describes each of the CPU registers.

Торі	c Page
3.1	CPU Primary Register File 3-2
3.2	CPU Expansion Register File 3-17

3.1 CPU Primary Register File

The 'C4x provides 32 registers in a multiport register file that is tightly coupled to the CPU. **The PC (program counter) is not included in the register file.** The contents of the register file are listed in Table 3–1.

Table 3–1. CPU Primary Register File

Register	Register Machine			
Symbol	Value (hex)	Assigned Function Name	Subsection	Page
R0	00	Extended-precision register 0	3.1.1	3-3
R1	01	Extended-precision register 1	3.1.1	3-3
R2	02	Extended-precision register 2	3.1.1	3-3
R3	03	Extended-precision register 3	3.1.1	3-3
R4	04	Extended-precision register 4	3.1.1	3-3
R5	05	Extended-precision register 5	3.1.1	3-3
R6	06	Extended-precision register 6	3.1.1	3-3
R7	07	Extended-precision register 7	3.1.1	3-3
R8	1C	Extended-precision register 8	3.1.1	3-3
R9	1D	Extended-precision register 9	3.1.1	3-3
R10	1E	Extended-precision register 10	3.1.1	3-3
R11	1F	Extended-precision register 11	3.1.1	3-3
AR0	08	Auxiliary register 0	3.1.2	3-4
AR1	09	Auxiliary register 1	3.1.2	3-4
AR2	0A	Auxiliary register 2	3.1.2	3-4
AR3	0B	Auxiliary register 3	3.1.2	3-4
AR4	0C	Auxiliary register 4	3.1.2	3-4
AR5	0D	Auxiliary register 5	3.1.2	3-4
AR6	0E	Auxiliary register 6	3.1.2	3-4
AR7	0F	Auxiliary register 7	3.1.2	3-4
DP	10	Data-page pointer	3.1.3	3-4
IR0	11	Index register 0	3.1.4	3-4
IR1	12	Index register 1	3.1.4	3-4
BK	13	Block-size register	3.1.5	3-5
SP	14	System stack pointer	3.1.6	3-5

Register Symbol	Register Machine Value (hex)	Assigned Function Name	See Subsection	On Page
ST	15	Status register		
DIE	16	DMA coprocessor interrupt en-	3.1.7	3-5
		able	3.1.8	3-8
IIE	17	Internal-interrupt enable register	3.1.9	3-11
lif	18	IIOF flag register (IIOF3–0 pins, timers, DMA)	3.1.10	3-13
RS	19	Repeat start address	3.1.11	3-16
RE	1A	Repeat end address	3.1.11	3-16
RC	1B	Repeat counter	3.1.11	3-16

	Table 3–1.	CPU Primary	Reaister File	(Continued
--	------------	-------------	---------------	------------

All of these registers can be used both as operands by the multiplier and ALU, and as general-purpose 32-bit registers. However, the registers also perform some special functions. For example, the 12 extended-precision registers maintain extended-precision floating-point results. The eight auxiliary registers support a variety of indirect addressing modes and can be used as general-purpose 32-bit integer and logical registers. The remaining registers provide system functions such as addressing, stack management, processor status, interrupts, and block repeat. Refer to Chapter 6, *Addressing Modes*, for detailed information and examples of how CPU registers are used in addressing.

3.1.1 Extended-Precision Registers (R0–R11)

The 12 extended-precision registers (R0–R11) can store and support operations on 32-bit integer and 40-bit floating-point numbers.

For floating-point numbers, these registers consist of two separate and distinct fields:

Bits 39–32: store the exponent (e) of a floating-point number.

Bits 31–0: store the mantissa of a floating-point number:

- Bit 31: sign bit (s),
- Bits 30–0: the fraction (f).

Any instruction that assumes that the operands are floating-point numbers uses bits 39–0. Figure 3–1 illustrates the storage of 40-bit floating-point numbers in the extended-precision registers.

Figure 3–1. Extended-Precision Register Floating-Point Format



For integer operations, bits 31–0 of the extended-precision registers contain the integer (signed or unsigned). Any instruction that assumes that the operands are either signed or unsigned integers uses only bits 31–0. Bits 39–32 remain unchanged. This is true for all shift operations. The storage of 32-bit integers in the extended-precision registers is shown in Figure 3–2.

Figure 3–2. Extended-Precision Register Integer Format

39 3	2 31 0
unchanged	signed or unsigned integer

3.1.2 Auxiliary Registers (AR0–AR7)

The eight 32-bit auxiliary registers (AR0–AR7) can be accessed by the CPU and modified by the two auxiliary register arithmetic units (ARAUs). The primary function of the auxiliary registers is the generation of 32-bit addresses. However, they can also operate as loop counters in indirect addressing or as 32-bit general-purpose registers that can be modified by the multiplier and ALU. See Chapter 6, *Addressing Modes*, for detailed information and examples of the use of auxiliary registers in addressing.

3.1.3 Data-Page Pointer (DP)

The data-page pointer (DP) is a 32-bit register whose 16 LSBs are used by the direct addressing mode as a pointer to the page of data being addressed. Data pages are 64K words long with a total of 64K (65,536) pages. *Bits 31–16 are reserved; they are always read as zeros and should not be modified by writing to the register.* The DP can be loaded by using the LDP pseudoinstruction or the LDI instruction. Figure 6–1, on page 6-5, describes this register's functions.

3.1.4 Index Registers (IR0, IR1)

The 32-bit index registers (IR0 and IR1) are used by the auxiliary register arithmetic unit (ARAU) for indexing the address. IR0 is also used for bit-reversed addressing. See Chapter 6, *Addressing Modes*, for detailed information and examples of the use of index registers in addressing. Section 6.4, *Indirect Addressing*, on page 6-6, discusses and provides examples of using IR*n* in indirect addressing. Section 6.9, *Bit-Reversed Addressing*, on page 6-32, describes using IR*n* with bit-reversed addressing.

3.1.5 Block-Size Register (BK)

The 32-bit block-size register (BK) is used by the ARAU in circular addressing to specify the data block size (see Section 6.8, *Circular Addressing*, on page 6-27, for more information about the use of the BK register).

3.1.6 System Stack Pointer (SP)

The system stack pointer (SP) is a 32-bit register that contains the address of the top of the system stack. The SP always points to the last element pushed onto the stack. The SP is manipulated by interrupts, traps, calls, returns, and the PUSH, PUSHF, POP, and POPF instructions. Pushes and pops of the stack perform preincrement and postdecrement, respectively, on all 32 bits of the SP.

3.1.7 Status Register (ST)

The status register (ST) contains global information about the CPU's state. Typically, load, store, arithmetic, and logical operations affect the ST's condition flags. When the ST is loaded, the contents of the load instruction's source operand replace the ST's current contents, regardless of the state of any bit(s) in the source operand. Therefore, following an ST load, the contents of the ST are identical to the contents of the source operand. This allows the status register to be saved easily and restored. At system reset, 0 is written to the ST; after reset, the CF bit is set to 1. The format of the ST is shown in Figure 3–3. The text following the figure describes each field in the ST.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ХХ	ХХ	хх	хх	хх	хх	хх	хх	ХХ	хх	хх	хх	NMI bus	s grant	ХХ	ANALYSIS
R	R	R	R	R	R	R	R	R	R	R	R	R	/W	R	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SC	PGIE	GIE	CC	CE	CF	PCF	RM	OVM	LUF	LV	UF	Ν	Z	V	С
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 3–3. Status Register (ST)

NOTE: xx = reserved bit. R = read, W = write.

- C Carry-condition flag.
- V Overflow condition flag.
- Z Zero condition flag.
- N Negative condition flag.
- UF Floating-point underflow condition flag.
- LV Latched overflow condition flag.
- LUF Latched floating-point underflow condition flag.

OVM Overflow mode (OVM) flag. This flag affects only integer operations.

If OVM = 0, the overflow mode is turned off.

If OVM = 1, integer results overflowing in the positive direction are set to the most positive 32-bit twos-complement number (7FFF FFFFh), and integer results overflowing in the negative direction are set to the most negative 32-bit twos-complement number (8000 0000h).

Note that the functions of bits V and LV are independent of the setting of OVM.

- **RM Repeat mode (RM) flag.** If RM = 1, the PC is modified in either the repeatblock or repeat-single mode.
- **PCF Previous state of bit CF.** When a trap executes or an interrupt is taken, the CF bit is set to 1 and the PCF bit is set to the CF bit's previous value.

The RETI and RETID instructions, explained in chapter 14, *Assembly Language Instructions*, copy PCF to the CF bit.

CF Cache freeze (CF). Enables or disables updating of the cache.

Set CF = 1 to freeze the cache. If CF = 1 and CE = 1, fetches from the cache and cache clearing (CC = 1) are allowed, but modification of the cache contents is not allowed. At reset, this bit is cleared to zero; it is set to 1 after reset.

When CF = 0, the cache is automatically updated by instruction fetches from external memory and cache clearing (CC = 1) is allowed. Traps and interrupts set CF. The RETI and RETID instructions copy the PCF bit to the CF bit.

Table 3–2 summarizes the CE and CF bits.

CE Cache enable (CE). CE enables or disables the instruction cache.

Set CE = 1 to enable the cache, allowing the cache to be used according to the LRU (least recently used) cache algorithm.

Set CE = 0 to disable the cache, preventing cache modifications and fetches. Cache clearing (CC = 1) is allowed when CE = 0. At reset, 0 is written to CE.

CC Cache clear. CC = 1 invalidates all entries in the cache (contents not guaranteed). This bit is always cleared after it is written to and thus always read as 0. At reset, 0 is written to this bit. All cache P flags = 0 when cache is cleared.

Table 3–2. Summary of the CE and CF Bits

CE	CF	Effect
0	0	Cache not enabled
0	1	Cache not enabled
1	0	Cache enabled and not frozen
1	1	Cache enabled but frozen (cache read only)

GIE	Global interrupt enable. Enables or disables all maskable interrupts.				
	If $GIE = 1$, the CPU responds to any enabled interrupts.				
	If GIE = 0, the CPU does not respond to any enabled interrupts. This bit does not affect NMIs. The IDLE, LAT, RETI, RETID, and TRAP instructions affect this bit's value. GIE is cleared to 0 when a trap is executed or an interrupt is taken.				
PGIE	Previous state of bit GIE. When a trap executes or an interrupt is taken, bit GIE is cleared to 0. When this occurs, the PGIE bit is set to the GIE bit's value before the trap or interrupt. Note that the RETI <i>cond</i> and RETI <i>cond</i> D instructions copy PGIE to the GIE bit. At reset, this bit is cleared to 0.				
SET COND	This bit determines how condition flags (ST bits $0-6$) are set.				
(SC)	If SET COND = 0, condition flags are set if the operation's target is any extended-precision register (R0-R11). This setting makes the 'C4x similar to the 'C3x, regarding condition flag settings. This bit is cleared to 0 at reset.				
	If SET COND = 1, condition flags are set if the target of the operation is any register in the primary register files <i>except</i> the status register. Condition flags are always set when a CMPF, CMPI, CMPF3, CMPI3, TSTB, or TSTB3 instruction is executed, regardless of the value of SET COND.				
ANALYSIS	This read-only bit is used in analysis mode to provide state information for emulation.				
NMI bus	('C44 and 'C40 revision \geq 5.0 only)				
grant	The NMI bus-grant feature is useful in correcting communication-port errors when used with the communication-port software reset feature. If bit $19 = 1$ and bit $18 = 0$, an internal peripheral bus-grant signal is forced on the falling edge of \overline{NMI} . If \overline{NMI} is asserted when the peripheral bus is in a stall condition, the NMI breaks the pending cycle and then jumps to the NMI service routine. A stall condition may occur when writing to a full output FIFO, or when reading from an empty input FIFO.				

xx Reserved. Value undefined. These bits are read-only.

3.1.8 DMA Coprocessor Interrupt Enable Register (DIE)

The 32-bit DMA interrupt enable register (DIE) is broken into six subfields that determine which interrupts can be used to control the *synchronization* for each of the six DMA coprocessor channels. Synchronization controls when a DMA channel reads or writes. At reset, zeros are written to all register bits.

Each DMA channel looks not only at the DMA synchronous interrupts selected but also at the synchronization mode that the channel is currently using (see Table 11–3). The synchronization mode is specified by the SYNC MODE field in the DMA channel control registers located in the DMA coprocessor.

By using interrupt synchronization, each DMA channel can (for example) service a corresponding communication port. Note that *DMA*i *can be synchronized only to signals coming from communication port* i *(where* $0 \le i \le 5$). Also, each DMA channel can be synchronized to external interrupts and to the on-chip timers.

3.1.8.1 Unified Mode

Figure 3–4 shows the DMA interrupt enable register for unified mode. Table 3–3 summarizes the interrupt activity for each of the four possible combinations of DMA0 and DMA1 for unified mode. Table 3–4 summarizes the interrupts enabled by three-bit values in DMA2 through DMA5 for unified mode.

Figure 3–4. DMA	Interrupt Enable	Register Bit	[•] Functions for	r DMA Ur	nified Mode
0	1				

31	30	29	28	27	26	25	24	23	22	21	20
C	DMA5 Write DMA5 Read			DMA4 Write			DMA4 Read				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
19	18	17	16	15	14	13	12	11	10	9	8
DMA3 Write		C	DMA3 Read		DMA2 Write		DMA2 Read				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
7		6	5		4	3		2	1		0
	DMA1 Write DMA1 Read		DMA0 Write		DMA0 Read						
R/W		R/W	R/W		R/W	R/W		R/W	R/W		R/W

R = Read W = Write

	Interru	upt Enabled	at DMA0 o	or DMA1	
Bit Value (in DMA0 or DMA1)	DMA0 Read	DMA0 Write	DMA1 Read	DMA1 Write	Interrupt Source for DMA Synchronization
0 0†	None	None	None	None	
0 1‡	ICRDY0	OCRDY0	ICRDY1	OCRDY1	From communication port
1 0	IIOF0	IIOF1	IIOF2	IIOF3	From external pins IIOF0-IIOF3
1 1	TIM0	TIM0	TIM0	TIM0	From timer TIM0

Table 3–3. DMA Channels 0 and 1 (DMA0 and DMA1) Unified Mode Synchronization Interrupts

[†] DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used. [‡] This option is not available for DMA0 and DMA3 in the 'C44.

Table 3-4. DMA Channels 2 to 5 (DMA2 to DMA5) Unified Mode Synchronization Interrupts

Bit Value	Interrupt Enab	oled at DMA2–DMA5 [†]	Interrupt Source for DMA		
(in DMA2 to DMA5)	DMA <i>x</i> Read	DMA <i>x</i> Write	Synchronization		
0 0 0‡	None	None			
0 0 1§	ICRDY <i>x</i> †	OCRDY <i>x</i> †	From communication port		
0 1 0	IIOF0	IIOF0	From external pins IIOF0–IIOF3		
0 1 1	IIOF1	IIOF1			
100	IIOF2	IIOF2			
101	IIOF3	IIOF3			
1 1 0	TIM0	TIMO	From timers TIM0 and TIM1		
111	TIM1	TIM1			

[†] The *x* in DMA*x* is the DMA channel number, which is also the number for the corresponding ICRDY*x* and OCRDY*x* interrupts. For example, an 001₂ in both DMA2 READ and DMA5 WRITE would enable interrupts ICRDY2 and OCRDY5, respectively.

All other viable bit values (010₂ to 111₂) are the same (as shown in the table) for DMA2 through DMA5.

[‡] DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

§ This option is not available for DMA0 and DMA3 in the 'C44.

Note: DMA Coprocessor Uses Signals to Synchronize

The interrupts in Table 3–3 and Table 3–4 (ICRDY*x*, OCRDY*x*, TIM0, etc.) *are not vectored*. The DMA coprocessor uses these as signals to synchronize DMA coprocessor transfers. This process is explained in Section 11.10.
3.1.8.2 Split Mode

Figure 3–5 shows the DMA interrupt enable register for split mode. Table 3–5 summarizes the interrupt activity for each of the four possible combinations of DMA0 and DMA1 for split mode. Table 3–6 summarizes the interrupts enabled by three-bit values in DMA2 through DMA5 for split mode.

Figure 3–5. DMA Interrupt Enable Register Bit Functions for DMA Split Mode

31	30	29	28	27	26	25	24	23	22	21	20	
DMA5 Primary Write			DMA5	5 Auxiliary	/ Read	DMA	DMA4 Primary Write			DMA4 Auxiliary Read		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
19	18	17	16	15	14	13	12	11	10	9	8	
DMA	3 Primary	Write	DMA3 Auxiliary Read		DMA2 Primary Write			DMA2 Auxiliary Read				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
7		6	5		4	3		2	1		0	
DMA1 Primary Write		DMA1	Auxiliary	Read	DMA0	Primary	Write	DMA0	Auxiliary	Read		
R/W		R/W	R/W		R/W	R/W		R/W	R/W		R/W	

R = Read W = Write

Table 3–5. DMA Channels 0 and 1 (DMA0 and DMA1) Split-Mode Synchronization Interrupts

Rit Value	Inter	rupt Enable	d at DMA0 oi	DMA1		
(in DMA0 or DMA1)	DMA0 Auxiliary Read	DMA0 Primary Write	DMA1 Auxiliary Read	DMA1 Primary Write	Interrupt Source for DMA Synchronization	
0 0†	None	None	None	None		
0 1‡	ICRDY0	OCRDY0	ICRDY1	OCRDY1	From communication port	
1 0	IIOF0	IIOF1	IIOF2	IIOF3	From external pins IIOF0-IIOF3	
11	TIM0	TIM0	TIM0	TIM0	From timer TIM0	

[†]DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

[‡] This option is not available for DMA0 and DMA3 in the 'C44.

	Interrupt Enable	ed at DMA2–DMA5 [†]	
Bit Value (in DMA2 to DMA5)	DMA <i>x</i> Auxiliary Read [†]	DMA <i>x</i> Primary Write [†]	Interrupt Source for DMA Synchronization
0 0 0‡	None	None	
0 0 1§	ICRDY <i>x</i> †	OCRDY <i>x</i> †	From communication port
0 1 0	IIOF0	IIOF0	From external pins IIOF0-IIOF3
0 1 1	IIOF1	IIOF1	
100	IIOF2	IIOF2	
101	IIOF3	IIOF3	
1 1 0	TIM0	TIMO	From timers TIM0 and TIM1
1 1 1	TIM1	TIM1	

Table 3–6. DMA Channels 2 to 5 (DMA2 to DMA5) Split-Mode Synchronization Interrupts

[†] The *x* in DMA*x* is the DMA channel number, which is also the number for the corresponding ICRDY*x* and OCRDY*x* interrupts. For example, an 001₂ in both DMA2 READ and DMA5 WRITE would enable interrupts ICRDY2 and OCRDY5, respectively. All other viable bit values (010₂ to 111₂) are the same (as shown in the table) for DMA2 through DMA5.

‡ DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

§ This option is not available for DMA0 and DMA3 in the 'C44.

3.1.9 CPU Internal Interrupt Enable Register (IIE)

The 32-bit internal interrupt enable register, shown in Figure 3–6, enables/disables the following interrupts for the CPU:

Timers 0 and 1

□ For communication ports 0–5:

- Input-buffer full
- Input-buffer ready
- Output-buffer ready
- Output-buffer empty

□ DMA coprocessor channels 0–5

Figure 3–6 shows the IIE register bits. A 1 means the corresponding interrupt is enabled; a 0 indicates disabled. At reset, zeros are written to all register bits.

31	30	29	28	27	26	25	2	24	23	3	22	21
ETINT1	EDMA INT5	EDMA INT4	EDMA INT3	EDMA INT2	EDMA INT1	EDM/ INT0	A EC EMI	DC- PTY5	EO RD	C– Y5	EIC- RDY5	EIC– FULL5
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	/W	R/	W	R/W	R/W
20	19	18	17	16	15	14	13	12		11	10	9
EOC EMPTY4	EOC RDY4	EIC RDY4	EIC FULL4	EOC EMPTY3	EOC RDY3	EIC RDY3	EIC FULL3	EO0 EMPT	C Y2	EOC RDY2	EIC RDY2	EIC FULL2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/V	V	R/W	R/W	R/W
8		7	6	5	4		3	2		1		0
EOC EMPTY	1 EC	OC DY1	EIC RDY1	EIC FULL1	EOC EMPT	Yo	EOC RDY0	EI RD	2 70	FUL		ETINT0
R/W	R	/W	R/W	R/W	R/W	1	R/W	R/\	N	R/V	V	R/W

Figure 3–6. Internal Interrupt Enable Register (IIE)

R = Read, W = Write, R/W = Read/Write

Notes:

- 1) In the figure, the shaded boxes are reserved bits in the 'C44. Zero should be written to each of these bits.
- 2) The fields corresponding to each unit are separated by double lines.

The following are definitions for each of the bits in the IIE.

- **EICFULL***x* Comm. port *x* input-buffer full interrupt
- **EICRDY***x* Comm. port *x* input-buffer ready interrupt
- **EOCRDY***x* Comm. port *x* output-buffer ready interrupt
- **EOCEMPTY***x* Comm. port *x* output-buffer empty interrupt
- **EDMAINT***x* DMA coprocessor channel *x* interrupt
- ETINT0 Timer 0 interrupt
- ETINT1 Timer 1 interrupt

In each field label, the *x* represents a communication port number (0-5) or a DMA coprocessor channel number (0-5). For example, a 1 in bit 5 causes interrupts to be generated when communication port number 1's input buffer becomes full. Or, a 1 in bit 26 enables channel 1 of the DMA coprocessor to respond to interrupts. A 1 enables each interrupt; a 0 disables it.

3.1.10 IIOF Flag Register (IIF)

The IIF register controls the external interrupt pins $\overline{IIOF}(3-0)$. Use it to specify:

- Which IIOF pins are used for general-purpose I/O and which are used for interrupts
- Whether a general-purpose pin is input (read only) or output (read/write)
- U Whether an interrupt pin is for edge-triggered or level-triggered interrupts,
- U Whether an external interrupt is enabled or disabled

The IIF register also contains timer, DMA and NMI interrupt flags. Figure 3–7 shows the IIF register's bits. The text following the figure explains these bits in detail.

The IIF register bits can be read from or written to under software control. This provides access to the \overline{IIOFx} pins, which can be treated as general-purpose I/O or as interrupt pins. For example, if at the IIF register, FUNCx = 0 (I/O pin) and TYPEx = 1 (output pin), then by writing into the FLAGx bit, you can also write to the external pin \overline{IIOFx} . If FUNCx = 1 (interrupt pin), writing a 1 to the IIF register FLAGx bit has the same effect as an incoming interrupt received on the corresponding pin. Consequently, all interrupts can be triggered and/or cleared through software. Since the interrupt bits also can be read from, the interrupt pins can be polled in software when an interrupt-driven interface is not required.

Internal interrupts operate in a similar manner. In the IIF register, the bit corresponding to an internal interrupt (e.g., TINT0, TINT1) can be read from and written to through software. Writing a 1 sets the interrupt latch, and writing a 0 clears it. All internal interrupts are one H1/H3 cycle in length. Modify the IIF by using logic operations (AND, OR, etc.) as shown:

correct			incorrec	incorrect				
LDI	@MAS	SK , R0	LDI IIF,	R	1			
AND	R0,	IIF	AND @MAS	@MASK, F				
			LDI R1,	II	F			

Traps and interrupts are described briefly in Section 3.2, *CPU Expansion Register File*, on page 3-17, and in detail in Section 7.4, *Interrupts*, on page 7-15, and Section 7.5, *Traps*, on page 7-24.

FUNCx

TYPE*x*

Figure 3–7. Interru	ipt Flag	Reaister	(<i>IIF</i>)
			()

_	31	30	29	28	27	26	25	24	
- [TINT1	DMAINT5	DMAINT4	DMAINT3	DMAINT2	DMAINT1	DMAINT0	TINT0	
-	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
_	23	22	21	20	19	18	17	16	
- [хх	хх	хх	хх	xx	xx	хх	NMI	
_	R	R	R	R	R	R	R	R	
_	15	14	13	12	11	10	9	8	
	EIIOF3	FLAG3	TYPE3	FUNC3	EIIOF2	FLAG2	TYPE2	FUNC2	
-	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
_	7	6	5	4	3	2	1	0	
	EIIOF1	FLAG1	TYPE1	FUNC1	EIIOF0	FLAG0	TYPE0	FUNC0	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R = Read, W = Write, R/W = Read/Write									
	Mode of pin IIOF <i>x</i> . If FUNC <i>x</i> = 0, pin $\overline{\text{IIOF}x}$ is a general-purpose I/O (R/W) pin. If FUNC <i>x</i> = 1, pin $\overline{\text{IIOF}x}$ is an <i>interrupt pin.</i>								
	Type of	f function	n for pin	IIOF <i>x</i> .					

If pin \overline{IIOFx} is a general-purpose I/O pin (FUNCx = 0):

- TYPEx = 0 makes \overline{IIOFx} an input pin.
- TYPEx = 1 makes IIOFx an output pin
- If pin IIOF x is an interrupt pin (FUNC x = 1):
 - TYPEx = 0 makes IIOFx an edge-triggered latched interrupt,
 - TYPEx = 1 makes IIOFx a level-triggered unlatched interrupt.

FLAG*x* Flag for pin IIOF*x*.

f pin IIOF <i>x</i> is a general-	purpose input pin (FUNC $x = 0$, TYPE $x = 0$),
FLAG <i>x</i> = the value of	oin \overline{IIOFx} and is read only.

- If pin \overline{IIOFx} is a general-purpose output pin (FUNCx = 0, TYPEx = 1),
- FLAGx = the value on pin \overline{IIOFx} and is R/W.
- If pin \overline{IIOFx} is an interrupt pin (FUNCx = 1):

FLAGx = 0 if interrupt is not asserted.

FLAGx = 1 if interrupt is asserted.

If 0 (zero) is written to FLAG*x*, the corresponding interrupt is cleared unless an interrupt is on the same pin; in that case, the interrupt will remain set.

EIIFO*x* **Disable**/enable external interrupt. EIIOFx = 0 disables external interrupts at pin IIOFx.

EIIOFx = 1 enables external interrupts at pin IIOFx.

NMI	Nonmaskable Interrupt flag (NMI). The NMI interrupt (on the external \overline{NMI} pin) behaves like other interrupts, except that it cannot be masked (disabled) by the GIE bit (ST bit 13) or by writing to the NMI bit. It is temporarily masked during delayed branches and multicycle CPU operations. At reset, this bit is cleared. An asserted interrupt is cleared only by servicing the interrupt. NMI is a negative-going, edge-triggered, latched interrupt. It is read-only.
	Reading NMI as 0 indicates that the interrupt is not asserted.
	Reading NMI as 1 indicates that the interrupt is asserted.
Reserved	Reserved; read as zeros.
TINT0	Timer interrupt flags 0 and 1.
TINT1	Reading TINT x as 0 indicates that the timer interrupt is not asserted. Reading TINT x as 1 indicates that the timer interrupt is asserted. A zero written to this bit clears the interrupt unless the interrupt is asserted at the same time; in that case, the interrupt will be shown as asserted.
DMAINT <i>x</i>	Interrupt flag for DMA coprocessor channels 0 to 5. Reading DMAINT <i>x</i> as 0 indicates that the channel interrupt is not asserted. Reading DMAINT <i>x</i> as 1 indicates that the channel interrupt is asserted. A zero written to this bit clears the interrupt unless the interrupt is asserted at the same time; in that case, the interrupt is shown as asserted.
	Notes:
	1) Shaded IIF bits 0, 1, 2, 3 apply to pin IIOF0; shaded IIF bits 4, 5, 6, 7 apply to IIOF1, etc.

2) The *x* represents the corresponding \overline{IIOF} interrupt pin ($\overline{IIOF0-3}$)

3.1.11 Block-Repeat (RS, RE) and Repeat-Count (RC) Registers

The 32-bit repeat start address register (RS) contains the starting address of the block of program memory to be repeated when the CPU is operating in the repeat mode.

The 32-bit repeat end address register (RE) contains the ending address of the block of program memory to be repeated when the CPU is operating in the repeat mode.

Note:

If RE < RS, the block of program memory is not repeated, and the code does not loop backwards. However, the ST(RM) bit remains set to 1.

The repeat-count register (RC) is a 32-bit register that specifies the number of times a block of code is to be repeated when a block repeat is performed. If RC contains the number n, the loop is executed n + 1 times.

3.1.12 Program Counter (PC)

The program counter (PC) is a 32-bit register containing the address of the next instruction to fetch. While the program counter is not part of the CPU register file, it can be modified by the same instructions that modify the program flow.

3.1.13 Reserved Bits and Compatibility

To retain compatibility with future members of the 'C4x family of microprocessors, reserved bits that are read as zero must be written as zero. Reserved bits that have an undefined value **must not** have their current value modified. In other cases, maintain the reserved bits as specified.

3.2 CPU Expansion Register File

This expansion register file contains two special control registers:

- Interrupt-vector table pointer (IVTP)
- Trap-vector table pointer (TVTP)

Table 3–7. CPU E	xpansion Registers
------------------	--------------------

Assembler Syntax	Register Machine Value (Hex)	Function Name
IVTP	00	Interrupt-vector table pointer. Points to start of the interrupt-vector table.
TVTP	01	Trap-vector table pointer. Points to start of the trap-vector table.

Use the **LDEP instruction** to load (copy) an expansion register to a primary register (e.g., to any of the auxiliary registers AR0–AR7; see Table 3–1 on page 3-2). For example:

LDEP IVTP,AR5 ; IVTP contents to AR5

Likewise, use the **LDPE instruction** to load (copy) a primary register to an expansion register. Neither of these instructions affects the status register condition flags.

LDPE AR5, IVTP ; AR5 contents to IVTP

Note that both the interrupt-vector table and the trap-vector table are required to lie on a 512-word boundary; thus, the nine least significant bits of these pointers are zeros (i.e., $10\ 0000\ 0000\ 2 = 512 = 200$). Write only zeros to these bits (though the register forces these to zeros).

The 32-bit **IVTP register** points to (is essentially the base address for) the interrupt-vector table (IVT) in memory.

The 32-bit **TVTP register** is essentially the base address for the trap-vector table (TVT) in memory. This table contains the vectors for the TRAP instruction's 512-trap addresses (TRAP0–TRAP511).

The interrupt and trap vector tables can share the same 512-byte space in memory. In this configuration, you can place trap vectors where there are no interrupt vectors. For example, since interrupt vector 02Ch is unused, you could place a trap vector at IVTP + 02Ch (which is also TVTP + 02Ch if the tables overlap) and then call that trap by specifying 02Ch in the TRAP instruction.

At reset, IVTP and TVTP are both set to zero.

Chapter 4

Memory and the Instruction Cache

The 'C40 accesses a total memory space of 4G 32-bit words (16G bytes) of program, data, and I/O space; the 'C44 accesses a total memory space of 32M 32-bit words (128M bytes).

Two internal RAM blocks of $1K \times 32$ bits each (4K bytes) and an internal ROM block containing a bootloader permit two accesses per block in a single cycle.

A 128 \times 32-bit instruction cache allows code to be stored off-chip in slower, lower-cost memories without degrading performance. The cache also speeds data fetches to the same physical space as the program because it does not burden the bus with program instruction fetches.

This chapter describes the memory maps and the instruction cache.

Topic

4.1	Memory Map 4-2	
4.2	Peripheral Bus Memory Map 4-5	
4.3	Instruction Cache 4-10	

4.1 Memory Map

The 'C4x memory space of 4 gigawords ($4G \times 32$ bits where $1G = 2^{30}$) is shown in the memory maps in Figure 4–1 and Figure 4–2. The contents of the first segment of address space, at 0000 0000h to 000F FFFFh, is selected by the value of the ROM enable (ROMEN) pin:

- ROMEN = 1. Addresses 0000 0000h-0000 0FFFh are an on-chip ROM block (reserved for bootloader operations), and addresses 0000 1000h-000F FFFFh are reserved.
- ROMEN = 0. The on-chip (reserved) ROM is disabled, and addresses 0000 0000h-000F FFFFh are mapped to the local bus.

Memory starting at 0010 0000h is not affected by ROMEN. The following is a general summary of address ranges:

- O000 0000h-000F FFFFh: Can be local bus or on-chip (reserved) ROM, depending on the value of ROMEN. If ROMEN=0, these addresses are mapped to the local bus. If ROMEN=1, these addresses are mapped to the on-chip ROM.
- 0010 0000h-0010 00FFh: Internal peripherals (DMA Instructions coprocessor, communications ports, timers, etc.).

loaded from these 2 areas.

- **0010 0100h–002F F7FFh:** Reserved.
- 002F F800h-002F FBFFh: 1K RAM Block 0.
- 002F FC00h-002F FFFFh: 1K RAM Block 1.
- 0030 0000h-7FFF FFFFh: Local bus. These addresses are mapped to the local bus.
- 8000 0000h-0FFFF FFFFh: Global bus. These addresses are mapped to the global bus.

CPU data accesses and DMA accesses can be made from any unreserved part of the 'C4x memory map. Instruction fetches can take place from any unreserved area of the 'C4x memory map, except from the peripheral space (addresses 0010 0000h-0010 00FFh).

Note:

The 'C4x internal ROM is generally reserved for TI internal use only. However, for high-volume applications, you can request that TI install your code in the internal ROM.

Figure 4–1. 'C40 Memory Map



Memory and the Instruction Cache

4-3



Figure 4–2. 'C44 Memory Map

4.2 Peripheral Bus Memory Map

The peripheral bus memory map resides in addresses 0010 0000h– 0010 00FFh. Each peripheral requires a 16-word area. Figure 4–3 shows the locations of registers for each peripheral in the memory map.

Figure 4–3. Peripheral Memory Map

0010 0000h	Local and Global Port Control (16 words)
0010 000Fh	(See subsection 4.2.1 and Figure 4–4)
0010 0010h	Analysis Module Block Registers (16 words)
0010 001Fh	(See subsection 4.2.2)
0010 0020h	Timer 0 Registers (16 words)
0010 002Fh	(See subsection 4.2.3 and Figure 4–5)
0010 0030h	Timer 1 Registers (16 words)
0010 003Fh	(See subsection 4.2.3 and Figure 4–5)
0010 0040h	Communication Port 0 (16 words) ('C40 only)
0010 004Fh	(See subsection 4.2.4 and Figure 4–5)
0010 0050h	Communication Port 1 (16 words)
0010 005Fh	(See subsection 4.2.4 and Figure 4–5)
0010 0060h	Communication Port 2 (16 words)
0010 006Fh	(See subsection 4.2.4 and Figure 4–5)
0010 0070h	Communication Port 3 (16 words) ('C40 only)
0010 007Fh	(See subsection 4.2.4 and Figure 4–5)
0010 0080h	Communication Port 4 (16 words)
0010 008Fh	(See subsection 4.2.4 and Figure 4–5)
0010 0090h	Communication Port 5 (16 words)
0010 009Fh	(See subsection 4.2.4 and Figure 4–5)
0010 00A0h	DMA Coprocessor Channel 0 (16 words)
0010 00AFh	(See subsection 4.2.5 and Figure 4–6)
0010 00B0h	DMA Coprocessor Channel 1 (16 words)
0010 00BFh	(See subsection 4.2.5 and Figure 4–6)
0010 00C0h	DMA Coprocessor Channel 2 (16 words)
0010 00CFh	(See subsection 4.2.5 and Figure 4–6)
0010 00D0h	DMA Coprocessor Channel 3 (16 words)
0010 00DFh	(See subsection 4.2.5 and Figure 4–6)
0010 00E0h	DMA Coprocessor Channel 4 (16 words)
0010 00EFh	(See subsection 4.2.5 and Figure 4–6)
0010 00F0h	DMA Coprocessor Channel 5 (16 words)
0010 00FFh	(See subsection 4.2.5 and Figure 4–6)

Memory and the Instruction Cache 4-5

4.2.1 Local and Global Memory Interface Control Registers

These registers control the local and global memory interfaces. They occupy the first 16-word block of the peripheral bus memory map, shown in Figure 4–3. The registers themselves are shown in Figure 4–4. Chapter 9, *External Bus Operation*, covers the operation of these registers.

These registers define several settings:

- The page sizes used for the two strobes of each port
- Address ranges over which the strobes are active
- Wait states
- Other similar operations that compose the memory interfaces

Figure 4–4. Memory Interface Control Registers



4.2.2 Analysis Module Registers

The second lowest 16-word block in the peripheral bus memory map, as shown in Figure 4–3, contains part of the analysis module registers. These registers are reserved for emulation functions. The *TMS320C4x C Source Debugger User's Guide (literature number SPRU054)* describes the analysis module user interface provided by the 'C4x debugger.

4.2.3 Timer Registers

This group of registers occupies the 0010 0020h–0010 003Fh range in the peripheral bus memory map shown in Figure 4–3, on page 4-5. Timers and their registers are covered in detail in Chapter 13, *Timers*.



Figure 4–5. Timer Registers

Memory and the Instruction Cache 4-7

4.2.4 Communication Port Memory Map

Figure 4–6 illustrates the communication-port control registers (CPCR) and input and output FIFO buffers. This is the central group of registers in the peripheral bus memory map shown in Figure 4–4, on page 4-6. These registers are described in more detail in Chapter 12, *Communication Ports*.

Figure 4–6. Communication Port Memory Map



4.2.5 DMA Coprocessor Registers

The DMA registers (shown in Figure 4–7) are the bottom block of registers in the peripheral bus memory map (Figure 4–3 on page 4-5). These registers are described in Chapter 11, *The DMA Coprocessor*.

Figure 4–7. DMA Coprocessor Memory Map



Exploded View of Each Channel Register

010 00z0h	Control register x	1
010 00z1h	Source address x	
010 00z2h	Source address index x	
010 00z3h	Transfer counter x	DMA
010 00z4h	Destination address x	Ch.
010 00z5h	Destination address index x	
010 00z6h	Link pointer x	
010 00z7h	Auxiliary transfer counter x	
010 00z8h	Auxiliary link pointer x	

- x = channel number (e.g., x = 1 for channel 1, x = 2 for channel 2, etc.)
- z = corresponding hexadecimal digit for channel address (e.g., substitute an A for DMA channel 0, B for DMA channel 1, etc.)



4.3 Instruction Cache

The 128×32 -bit instruction cache speeds instruction fetches and lowers system cost. The instruction cache allows the use of slow external memories while still achieving single-cycle access performance. The cache also frees the external buses from program fetches, thus, allowing the use of these buses for DMA or other system needs. The cache can operate in a completely automatic fashion without the need for external intervention. It uses a form of the LRU (least recently used) cache update algorithm.

4.3.1 Instruction Cache Architecture

The instruction cache (see Figure 4–9 on page 4-11) contains 128 32-bit words of RAM, enough to hold 128 words of program memory. It is divided into four 32-word segments. Associated with each segment is a 27-bit segment start address (SSA) register. For each word in the cache, there is a corresponding single-bit present (P) flag.

When the CPU requests an instruction word, a check is made to determine whether the word is already in the instruction cache. The partitioning of an instruction address as used by the cache control algorithm is shown in Figure 4–8. The 27 most significant bits (MSBs) of the instruction address select the segment, and the five least significant bits (LSBs) define the address of the instruction word within the pertinent segment. The 27 MSBs of the instruction address are compared with the four SSA registers. If a match is found, the relevant P flag is checked. The P flag indicates whether the word within a particular segment is already present in cache memory:

- \square P = 1: the word is already present in cache memory.
- P = 0: location in cache is invalid (e.g., contains garbage).

Figure 4–8. Address Partitioning for Cache Control Algorithm

31		5 4	0
	Segment start address	Instruction	word
	(SSA)	address within	segment

If there is no match, one of the segments must be replaced by the new data. The segment replaced in this circumstance is determined by the LRU (least recently used) algorithm. The LRU stack (see the upper-right portion of Figure 4–9) is maintained for this purpose.





Memory and the Instruction Cache 4-11

The LRU stack keeps track of which segment (0-3) qualifies as the least recently used after each access to the cache. Each time a segment is accessed, its segment number is removed from the LRU stack and pushed onto the top of the LRU stack. Therefore, the number at the top of the stack is the most recently used segment number, and the number at the bottom of the stack is the least recently used segment number.

At reset, the following occur in the instruction cache:

- Cache is disabled (ST(CE) = 0). After reset cache is frozen (ST(CF) = 1). See section 3.1.7, *Status Register (ST)*, on page 3-5, for details.
- All P flags are set to zero.
- The LRU stack is initialized with segment 0 at the top, followed by segments 1, 2, and 3 at the bottom. If any two SSA registers are equal (due to reset conditions) and a cache hit occurs, the instruction word is fetched from the most recently used segment.

When a replacement is necessary, the least recently used segment is selected for replacement. Also, the 32 P flags for the segment to be replaced are set to 0, and the segment's SSA register is replaced with the 27 MSBs of the new instruction's address.

4.3.2 Cache Control Bits

Four cache control bits are located in the CPU status register (ST): the cache clear bit (CC), the cache enable bit (CE), the cache freeze bit (CF), and the previous cache freeze bit (PCF). The status register is shown in Figure 3–3.

Cache Clear Bit (CC). Set CC = 1 to invalidate all entries in the cache. This bit is always cleared after it is written to; thus, it is always read as 0. At reset, 0 is written to this bit. The cache P flag = 0 when the cache is cleared.

Cache Enable Bit (CE). Set CE = 1 to enable the cache, allowing the cache to be used according to the LRU (least recently used) cache algorithm. Set CE = 0 to disable the cache; this prevents cache updates or modifications (thus, no cache fetches can be made). At reset, 0 is written to this bit. Cache clearing (CC = 1) is allowed when CE = 0.

Cache Freeze Bit (CF). Set CF = 1 to freeze the cache including freezing of LRU (least recently used) stack manipulation. If the cache is enabled (CE = 1) and the cache is frozen (CF = 1), fetches from the cache are allowed, but modification of the cache contents is not allowed. Cache clearing (CC = 1) is allowed when CF = 1. At reset, this bit is cleared to 0 and after reset it is set to 1. When CF = 0, cache clearing (CC=1) is allowed. CF is set to one when a trap or interrupt is taken. Also, the RETI and RETID instructions copy PCF to the CF bit.

Table 4–1 summarizes the effects of the CE and CF bits.

Table 4–1. Combined Effect of the CE and CF Bits

CE	CF	Effect
0	0	Cache not enabled
0	1	Cache not enabled
1	0	Cache enabled and not frozen
1	1	Cache enabled and frozen

Previous Cache Freeze Bit (PCF). When an interrupt or trap vector is taken, the CF value is copied to the PCF bit, and the CF bit is set to 1. This protects the cache during interrupt processing and is particularly useful when code loops are interrupted. The interrupt service routine may optionally use the cache under software control. Interrupts may also be nested, providing that the status register is saved before the interrupts are enabled. When the instructions RETI*cond* and RETI*cond*D are executed to complete interrupt processing, the contents of the PCF bit are copied to the CF bit.

4.3.3 Using the Cache

Only instructions may be fetched from the program cache. All reads and writes of data to and from memory, bypass the cache. Program fetches from internal memory do not modify the cache and do not generate cache hits or misses. The program cache is a single-access memory block. Dummy program fetches (i.e., following a branch) can generate cache misses and cache updates. Example 4–1 shows a typical way to clear and enable the cache.

Example 4–1. Enabling the Cache

OR 1800h, ST

To use the cache more efficiently, take two precautions:

Avoid using self-modifying code. If an instruction resides in the cache and the corresponding location in primary memory is modified, the copy in the instruction in the cache is not modified.

Align program code. Use the *.align* directive when coding assembly language to align code on 32-word address boundaries.

4.3.4 The LRU Cache Algorithm

When the 'C4x requests an instruction word from external memory, the two possible actions are a *cache hit* or a *cache miss*:

- **Cache Hit.** The cache contains the requested instruction, and the following actions occur:
 - The instruction word is read from the cache.
 - The number of the segment containing the word is removed from the LRU stack and pushed to the top of the LRU stack (if it is not already at the top), thus moving the other segment numbers toward the bottom of the stack.
- Cache Miss. The cache does not contain the instruction. There are two types of cache misses:
 - Subsegment miss. The segment address register matches the instruction address, but the relevant P flag is not set. The following actions occur:
 - The instruction word is read from memory and copied into the cache.
 - The number of the segment containing the word is removed from the LRU stack and pushed to the top of the LRU stack (if it is not already at the top), thus moving the other segment numbers toward the bottom of the stack.
 - The relevant P flag is set.
 - Segment miss. None of the segment addresses matches the instruction address. The following actions occur:
 - The least recently used segment is selected for replacement and the P flags for all 32 words are cleared.
 - The SSA register for the selected segment is loaded with the 27 MSBs of the address of the requested instruction word.
 - The instruction word is fetched and copied into the cache. It goes into the appropriate word of the least recently used segment. The P flag for that word is set to 1.
 - The number of the segment containing the instruction word is removed from the LRU stack and pushed to the top of the LRU stack, thus moving the other segment numbers toward the bottom of the stack.

Chapter 5

Data Formats and Floating-Point Operation

In the 'C4x architecture, data is organized into three fundamental types: integer, unsigned-integer, and floating-point. Note that the terms, integer and signed-integer, are considered to be equivalent. The 'C4x supports short and single-precision formats for signed and unsigned integers. It also supports short, single-precision and extended-precision formats for floating-point data.

Floating-point operations make fast, trouble-free, accurate, and precise computations. Specifically, the 'C4x implementation of floating- point arithmetic facilitates floating-point operations at integer speeds while preventing problems with overflow, operand alignment, and other burdensome tasks common in integer operations.

This chapter discusses in detail the data formats and floating-point operations supported on the 'C4x.

Topic

5.1	Signed-Integer Formats 5-2
5.2	Unsigned-Integer Formats 5-3
5.3	Floating-Point Formats 5-4
5.4	Floating-Point Conversion (IEEE Std. 754) 5-13
5.5	Floating-Point Multiplication 5-19
5.6	Floating-Point Addition and Subtraction 5-23
5.7	Normalization (NORM Instruction) 5-27
5.8	Rounding (RND Instruction)
5.9	Floating-Point to-Integer Conversion (FIX Instruction)
5.10	Integer-to-Floating-Point Conversion (FLOAT Instruction) 5-33
5.11	Reciprocal (RCPF Instruction) 5-34
5.12	Reciprocal Square Root (RSQRF Instruction) 5-36

Page

5.1 Signed-Integer Formats

The 'C4x supports two signed-integer formats: a 16-bit short format and a 32-bit single-precision format. The term *integer* is used throughout this chapter to refer to a signed integer.

Note:

When extended-precision registers are used as integer operands, only bits 31–0 are used; bits 39–32 remain unchanged and unused.

5.1.1 Short Integer Format

The 16-bit twos-complement short integer format is used for immediate integer operands. For those instructions that assume integer operands, this format is sign extended to 32 bits (see Figure 5–1). The range of an integer *si*, represented in the short integer format, is:

 $-2^{15} \le si \le 2^{15} -1$

In Figure 5–1 and other figures in this chapter, s = sign bit.

Figure 5–1. Short-Integer Format and Sign Extension of Short Integer



(b) Sign extension of a short integer format

5.1.2 Single-Precision Integer Format

In the single-precision integer format, the integer is represented in twos-complement notation. The range of an integer *sp*, represented in the single-precision integer format, is $-2^{31} \le sp \le 2^{31} - 1$. Figure 5–2 shows the single-precision integer format.





5-2

5.2 Unsigned-Integer Formats

Two unsigned-integer formats are supported on the 'C4x: a 16-bit short format and a 32-bit single-precision format. In this chapter, the term *unsigned integer* is used to refer to an unsigned integer.

Note:

When extended-precision registers are used, the unsigned-integer operands use only bits 31–0; bits 39–32 remain unchanged.

5.2.1 Short Unsigned-Integer Format

Figure 5–3 shows the 16-bit short unsigned-integer format used in immediate unsigned-integer operands. For instructions that use unsigned-integer operands, the format is filled with zeros to 32 bits. The range of a short unsigned integer is $0 \le si \le 2^{16}$.

Figure 5–3. Short Unsigned-Integer Format and Zero Fill



(b) Zero fill of a short unsigned-integer format

5.2.2 Single-Precision Unsigned-Integer Format

In the single-precision unsigned-integer format, the number is represented as a 32-bit value, as shown in Figure 5–4. The range of a single-precision unsigned-integer is $0 \le sp \le 2^{32}$.

Figure 5–4. Single-Precision Unsigned-Integer Format

31 0

5.3 Floating-Point Formats

The 'C4x supports three floating-point formats:

- A short floating-point format (for immediate floating-point operands) consisting of a 4-bit exponent, one sign bit, and an 11-bit fraction
- A single-precision format consisting of an 8-bit exponent, one sign bit, and a 23-bit fraction
- An extended-precision format consisting of an 8-bit exponent, one sign bit, and a 31-bit fraction

All 'C4x floating-point formats consist of three fields: *an exponent field* (e), *a single-bit sign field* (s), *and a fraction field* (f). The sign field and fraction field may be considered as one unit and referred to as the *mantissa field* (man). Each format is divided into these fields as shown in Figure 5–5.

Figure 5–5. General Floating-Point Format



The general equation for calculating the value in a floating point number is given by Equation 5–1. In the equation, s is the value of the sign bit, \overline{s} is the inverse of the value of the sign bit, f is the binary value of the fraction field, and e is the decimal equivalent of the exponent field.

Equation 5–1. Value in a Floating Point Number

$$x = s\overline{s}.f_2 \times 2^{e}$$

The mantissa represents a normalized twos-complement number. In a normalized representation, a most significant nonsign bit is implied, thus providing an additional bit of precision. The implied sign bit is used as follows:

- If s = 0, then the leading two bits of the mantissa are 01.
- \Box If s = 1, then the leading two bits of the mantissa are 10.

If the sign bit, s, is equal to 0, the mantissa becomes $01.f_2$, where f is the binary representation of the fraction field. If s is 1, the mantissa becomes $10.f_2$, where f is the binary representation of the fraction field.

For example, if $f = 0000000001_2$ and s = 0, the value of the mantissa (*man*) would be 01.0000000001_2. If s = 1 for the same value of f, the value of *man* would be 10.0000000001_2.

The exponent field is a twos-complement number that determines the factor of two by which the number is multiplied. Essentially, the exponent field shifts the binary point in the mantissa. If the exponent is positive, then the binary point is shifted to the right. If the exponent is negative, then the binary point is shifted to the left.

For example, if $man = 01.0000000001_2$ and the $e = 11_{10}$, then the binary point is shifted eleven places to the right, producing the number: 01000000001_2 , which is equal to 2049 decimal.

5.3.1 Short Floating-Point Format

In the short floating-point format, floating-point numbers are represented by a twos-complement 4-bit exponent field (*e*) and a twos-complement 12-bit mantissa field (*man*) with an implied most significant nonsign bit.





You must use the following reserved values to represent zero in the single-precision floating-point format:

$$e = -8$$
$$s = 0$$
$$f = 0$$

Operations are performed with an implied binary point between bits 11 and 10. The floating-point twos-complement number x in the short floating-point format is given by:

$x = 01.f_2 \times 2^e$	if <i>s</i> = 0
$x = 10.f_2 \times 2^{e}$	if <i>s</i> = 1
<i>x</i> = 0	if $e = -8$, $s = 0$, $f = 0$

The following examples illustrate the range and precision of the short floating-point format:

Most Positive:	$x = (2 - 2^{-11}) \times 2^7 = 2.5594 \times 10^2$
Least Positive:	$x = 1 \times 2^{-7} = 7.8125 \times 10^{-3}$
Least Negative:	$x = (-1 - 2^{-11}) \times 2^{-7} = -7.8163 \times 10^{-3}$
Most Negative:	$x = -2 \times 2^7 = -2.5600 \times 10^{2}$

Data Formats and Floating-Point Operation 5-5

5.3.2 Single-Precision Floating-Point Format

In the single-precision format, the floating-point number is represented by an 8-bit exponent field (*e*) and a twos-complement 24-bit mantissa field (*man*) with an implied most significant nonsign bit.

Operations are performed with an implied binary point between bits 23 and 22. When the implied most significant nonsign bit is made explicit, it is located to the immediate left of the binary point. The floating-point number x is given by

$x = 01.f \times 2^{e}$	if $s = 0$
$x = 10.f \times 2^{e}$	if <i>s</i> = 1
<i>x</i> = 0	if <i>e</i> = −128, <i>s</i> = 0, f = 0





You must use the following reserved values to represent zero in the single-precision floating-point format:

e = -128 *s* = 0

f = 0

The following examples illustrate the range and precision of the single-precision floating-point format.

Most Positive:	$x = (2 - 2^{-23}) \times 2^{127} = 3.4028234 \times 10^{38}$
Least Positive:	$x = 1 \times 2^{-127} = 5.8774717 \times 10^{-39}$
Least Negative:	$x = (-1 - 2^{-23}) \times 2^{-127} = -5.8774724 \times 10^{-39}$
Most Negative:	$x = -2 \times 2^{127} = -3.4028236 \times 10^{38}$

5.3.3 Extended-Precision Floating-Point Format

In the extended-precision format, the floating-point number is represented by an 8-bit exponent field (*e*) and a 32-bit mantissa field (*man*) with an implied most significant nonsign bit.

Operations are performed with an implied binary point between bits 31 and 30. When the implied most significant nonsign bit is made explicit, it is located to the immediate left of the binary point. The floating-point number x is given by:

$x = 01.f \times 2^{e}$	if <i>s</i> = 0
x =10.f × 2 ^e	if <i>s</i> = 1
<i>x</i> = 0	if <i>e</i> = −128, <i>s</i> = 0, <i>f</i> = 0





You must use the following reserved values to represent zero in the extended-precision floating-point format:

e = -128s = 0f = 0

The following examples illustrate the range and precision of the extended-precision floating-point format:

Most Positive:	$x = (2 - 2^{-31}) \times 2^{127} = 3.4028236683 \times 10^{38}$
Least Positive:	$x = 1 \times 2^{-127} = 5.8774717541 \times 10^{-39}$
Least Negative:	$x = (-1 - 2^{-31}) \times 2^{-127} = -5.8774717569 \times 10^{-39}$
Most Negative:	$x = -2 \times 2^{127} = -3.4028236691 \times 10^{38}$

5.3.4 Determining the Decimal Equivalent of a Floating-Point Number

There are two basic steps in determining the value stored in floating point format:

- 1) Determine the values of the exponent and mantissa.
- Shift the binary point in the mantissa according to the value of the exponent field and then convert the number to decimal.

5.3.4.1 Step 1: Determine the Values of the Exponent and Mantissa

The exponent field is a twos-complement number whose range depends on the type of floating-point number you are converting. Record the decimal equivalent of this value as *e*.

For example, if you are converting a single-precision floating-point number and the binary value of the exponent field is 00000100, then the decimal value of the exponent would be 4 since a 1 in the third bit from the right corresponds to 4.

If, on the other hand, the binary value of the exponent field is 11111100_2 , then the decimal value of the exponent would be -4. Since the first bit on the left is 1, you know that the number is negative. You calculate the value of the number by taking the one's complement of 11111100_2 , which is 00000011_2 and then by adding 1 to that result.

Note:

If the value of the exponent matches the value reserved for zero, then the floating point number is equal to zero. The reserved value for each floating point type is given with the type descriptions in Section 5.3.

The mantissa is a binary number with an implied binary point between the sign bit and the fraction field. Form the mantissa in one of two ways:

If s = 0, form the mantissa by writing 01. and appending the bits in the fraction field after the binary point.

For example, if *f* = 1010000000₂, then *man* = 01.1010000000₂:

	S			Fraction									
ſ	0].	1	0	1	0	0	0	0	0	0	0	0

Rewrite the mantissa as:



If s = 1, form the mantissa by writing 10. and appending the bits in the fraction field after the binary point.

For example, if $f = 1010000000_2$, then $man = 10.1010000000_2$.



Rewrite the mantissa as:

Mantissa														
1		0		1	0	1	0	0	0	0	0	0	0	0

5.3.4.2 Step 2: Shift the Decimal Point in the Mantissa and Convert to Decimal

If the exponent (e) has a positive value, then you shift the binary point e places to the right.

If the exponent (e) has a negative value, then you shift the binary point *e* places to the left.

For example, if $e = 2_{10}$ and the $man = 01.1100000000_2$, then the shifted mantissa becomes 0111.00000000_2 , which is equivalent to 7 in decimal.

If, on the other hand, $e = -2_{10}$ and $man = 01.1000000000_2$, then the shifted mantissa becomes .011000000000_2, which is equivalent to 3/8 in decimal.

The following examples illustrate how you can obtain the equivalent floatingpoint value of a number in 'C4x floating-point format. Each of the examples uses the single-precision floating point format.

Example 5–1. Positive Number

0 2 4 0 0 0 0 0 Hex value 0000 0010 0100 0000 0000 0000 0000 0000 Binary value Exponent = $0000 \ 0010_2 = 2$ Sign 0 .100002 Fraction = $01.1_2 \times 2^2 = 0110_2. = 6$ Value - Fraction Implied Sign

Data Formats and Floating-Point Operation 5-9

Example 5–2. Negative Number

Example 5–3. Fractional Number



5.3.5 Conversion Between Floating-Point Formats

Floating-point operations assume several different formats for inputs and outputs. These formats often require conversion from one floating-point format to another (for example, from short floating-point format to extended-precision floating-point format). Format conversions occur automatically in hardware, with no overhead, as a part of floating-point operations. Examples of the four conversions are shown in Figure 5–9 through Figure 5–12 (s = sign bit of the exponent). When a floating-point format zero is converted to a different format, it is always converted to a valid representation of zero in that format.

Figure 5–9. Short Floating-Point Format Conversion to Single-Precision Floating-Point Format



In converting from short format to single-precision format, the exponent field is sign extended and the rightmost 12 bits of the fraction field are filled with zeros.

Figure 5–10. Short Floating-Point Format Conversion to Extended-Precision Floating-Point Format



(b) Extended-precision floating-point format

In converting from short format to extended-precision format, the exponent field is sign extended and the rightmost 20 bits of the fraction field are filled with zeros.

Data Formats and Floating-Point Operation 5-11



			31	24	1 23	22				0
			x	X	s	у				У
		_		(a) S	ingle	e-precisio	n floating	-pior	nt format	
39	32 3	31 30)					8	7	0
x	x	s y						у	0	0

(b) Extended-precision floating-point format

In converting from single-precision format to extended-precision format, the rightmost eight bits of the fraction field are filled with zeros.

Figure 5–12. Extended-Precision Floating-Point Format Conversion to Single-Precision Floating-Point Format

39 32 3	31 30	0			8	7	0
x x s	s y	,			У	Ζ	Ζ
		31	24	23	22		0
		x	x	s	у		y

(b) Single-precision floating-point format

In converting from extended-precision format to single-precision format, the eight rightmost bits of the fraction field are truncated.

5.4 Floating-Point Conversion (IEEE Std. 754)

The 'C4x floating-point format is not compatible with the IEEE standard 754 format. However, the 'C4x has instructions to directly convert to and from IEEE format (TOIEEE and FRIEEE, respectively). The conversion process is explained in subsections 5.4.1 and 5.4.2. Figure 5–13 shows the IEEE floating-point format, and Figure 5–14 shows the floating-point 'C4x format.

Figure 5–13. IEEE Single-Precision Std. 754 Floating-Point Format



The following five cases define the value *v* of a number expressed in the IEEE format:

1)	lf	<i>e</i> = 255	and $f \neq 0$,	then	v = NaN
2)	lf	<i>e</i> = 255	and $f = 0$,	then	$v = (-1)^{s}$ infinite
3)	lf	0 < <i>e</i> < 25	55,	then	$v = (-1)^{s} \times 2^{e-127}(1.f)$
4)	lf	<i>e</i> = 0	and $f \neq 0$,	then	$v = (-1)^{s} \times 2^{-126}(0.f)$
5)	lf	<i>e</i> = 0	and $f = 0$,	then	$v = (-1)^{s} \times 0$ (zero).

where s = sign bit; e = the exponent field; f = the fraction field; NaN = Not a number

For the above five representations, *e* is treated as an unsigned integer. Case 1 generates NaN (not an number) and is primarily used for software signaling. Case 4 represents a denormalized number. Case 5 represents positive and negative zero.

Figure 5–14. 'C4x Single-Precision Twos-Complement Floating-Point Format[‡]



[‡]Same format as for the 'C3x

In comparison, Figure 5–14 shows the the 'C4x twos-complement floatingpoint format. In this format, two cases can be used to define value v of a number:

1)	lf	<i>e</i> = -128	and	<i>f</i> ≠ 0,	then $v = 0$
2)	lf	<i>e</i> ≠ −128			then $v = s\overline{s}.f_2 \times 2\epsilon$

where s = sign bit; *e* = the exponent field; *f* = the fraction field.

Data Formats and Floating-Point Operation 5-13
For this representation, *e* is treated as a twos-complement integer. The fraction and sign bit form a normalized twos-complement mantissa.

Note: Differentiating Symbols for IEEE and 'C4x Formats

To differentiate between the symbols that define these two formats, all IEEE fields are subscripted with an IEEE (e.g., e_{IEEE} , s_{IEEE} , etc.). Similarly, all twos-complement fields are subscripted with two (i.e., e_{two} , s_{two} , f_{two}).

5.4.1 Converting IEEE Format to Twos-Complement 'C4x Floating-Point Format

The most common conversion is the IEEE-to-twos-complement format. This conversion is done according to rules in the following table:

Table 5–1. Converting IEEE Format to Twos-Complement Floating-Point Format

	If These Values Are Present			Then These Values Equal			
Case	e _{IEEE}	SIEEE	f _{IEEE}	e _{two}	s _{two}	f _{two}	SIEEE
1	255	1		7Fh	1	00 0000h	
2	255	0		7Fh	0	7F FFFFh	
3	0< e _{IEEE} <255	0		e _{IEEE} –7Fh		fieee	0
4	0< e _{IEEE} <255	1	≠0	e _{IEEE} –7Fh		f _{IEEE} +1 [†]	1
5	0< e _{IEEE} <255	1	0	e _{IEEE} –80h		0	1
6	0			80h	0	00 0000h	

 $\dagger \overline{f}_{IEEE}$ = ones complement of f_{IEEE}.

Case 1 maps the IEEE positive NaNs and positive infinity to the single-precision twos-complement most positive number. Overflow is also signaled to allow you to check for these special cases.

Case 2 maps the IEEE negative NaNs and negative infinity to the singleprecision twos-complement most negative number. Overflow is also signaled to allow you to check for these special cases.

Case 3 maps the IEEE positive normalized numbers to the identical value in the twos-complement positive number.

Case 4 maps the IEEE negative normalized numbers with a nonzero fraction to the identical value in the twos-complement negative number.

Case 5 maps the IEEE negative normalized numbers with a zero fraction to the identical value in the twos-complement negative number.

Case 6 maps the IEEE positive and negative denormalized numbers and positive and negative zeros to a twos-complement zero.

The 'C4x assumes that an IEEE number is stored as an integer in memory or in a register. When the 'C4x converts an IEEE number, it places the number in an extended-precision register by using the exponent and fraction fields of the register. The eight LSBs of the extended-precision register are set to zero. Any arithmetic operations that are performed on the fraction field of the IEEE number should be performed only on the IEEE fraction field. In the case of a block memory transfer, a no-penalty data format conversion can be executed by using parallel instructions with STF. Example 5–4 illustrates how this can be accomplished.

Example 5–4. IEEE to 'C4x Conversion Within Block Memory Transfer

*	TITLE IEEE TO 'C4x CONVERSION WITHIN BLOCK MEMORY
*	TRANSFER
*	
*	PROGRAM ASSUMES THAT INPUT FIFO OF COMMUNICATION PORT 0
*	IS FULL OF IEEE FORMAT DATA. EIGHT DATA WORDS ARE
*	TRANSFERRED FROM COMMUNICATION PORT 0 TO INTERNAL RAM
*	BLOCK 0 AND THE DATA FORMAT IS CONVERTED FROM IEEE FORMAT
*	TO 'C4x FLOATING-POINT FORMAT.
*	
	LDI @CP0_IN,AR0 ;Load comm port0 input FIFO address
	LDI @RAM0,AR1 ;Load internal RAM block 0 address
	<pre>FRIEEE *AR0,R0 ;Convert first data</pre>
	RPTS 6
	<pre>FRIEEE *AR0,R0 ;Convert next data</pre>
	<pre>STF R0,*AR1++(1) ;Store previous data</pre>
	<pre>STF R0,*AR1++(1) ;Store last data</pre>

5.4.2 Converting Twos-Complement 'C4x Floating-Point Format to IEEE Format

This conversion is performed according to the following table:

Table 5–2. Converting Twos-Complement Floating-Point Format to IEEE Format

	If These Val	ues Are Preser	nt	Then These Values Equal		
Case	e _{two}	s _{two}	f _{two}	e _{IEEE}	SIEEE	f _{IEEE}
1	–128			00h	0	00 0000h
2	-127			00h	0	00 0000h
3	–126≤ <i>e_{two} ≤</i> 127	0		<i>e_{two}</i> +7Fh	0	f _{two}
4	–126≤ <i>e_{two}</i> ≤127	1	≠0	<i>e_{two}</i> +7Fh	0	\overline{f}_{two} +1†
5	–126≤ <i>e_{two}</i> ≤127	1	0	<i>e_{two}</i> +80h	1	00 0000h
6	127	1	0	FFh	1	00 0000h

† f_{two} = ones complement of f_{two} .

Case 1 maps a twos-complement zero to a positive IEEE zero.

Case 2 maps the twos-complement numbers that are too small to be represented as normalized IEEE numbers to a positive IEEE zero.

Case 3 maps the positive twos-complement numbers that are not covered by case 2 into the identically valued IEEE number.

Case 4 maps the negative twos-complement numbers with a nonzero fraction that are not covered in case 2 into the identically valued IEEE number.

Case 5 maps all the negative twos-complement numbers with a zero fraction, except for the most negative twos-complement number and those that are not covered in case 2, into the identically valued IEEE number.

Case 6 maps the most negative twos-complement number to the IEEE negative infinity.

The 'C4x assumes that the twos-complement numbers are in memory or are in an extended-precision register in the exponent and fraction field of the register (shown in Figure 5–14 on page 5-13). If the value is in an extended-precision register, then only the 24 MSBs of the fraction field are manipulated as the fraction field and for detection of the special cases. The result of the conversion goes into the 32 MSBs of an extended-precision register. In the case of a block memory transfer, a no-penalty data format conversion can be executed by using parallel instructions with STF. Example 5–5 illustrates how this can be accomplished.

```
Example 5–5. 'C4x to IEEE Conversion Within Block Memory Transfer
```

*	TITLE 'C4x TO IEEE CONVERSION WITHIN BLOCK MEMORY
*	TRANSFER
*	
*	PROGRAM ASSUMES THAT OUTPUT FIFO OF COMMUNICATION PORT 0
*	IS EMPTY. EIGHT DATA WORDS ARE TRANSFERRED FROM
*	INTERNAL RAM BLOCK 0 TO COMMUNICATION PORT 0 AND THE
*	DATA FORMAT IS CONVERTED FROM 'C4x FLOATING-POINT FORMAT
*	TO IEEE FORMAT.
*	
	LDI @CP0_OUT,AR0;Load comm port0 output FIF0 ; address
	LDI @RAM0,AR1 ;Load internal RAM block 0 ; address
	TOIEEE *AR1++(1),R0;Convert first data RPTS 6
	TOIEEE *AR1++(1),R0 ;Convert next data
	STF R0,*AR0 ;Store previous data
	STF R0,*AR0 ;Store last data

5.5 Floating-Point Multiplication

A floating-point number α can be written in floating-point format as in the following formula, where $\alpha(man)$ is the mantissa and $\alpha(exp)$ is the exponent: $\alpha = \alpha(man) \times 2^{\alpha(exp)}$

The product of α and b is c, defined as:

 $c = \alpha \times b = \alpha(man) \times b(man) \times 2^{(\alpha(exp)+b (exp))}$

Thus:

 $c(man) = \alpha(man) \times b(man)$

 $c(exp) = \alpha(exp) + b(exp)$

During floating-point multiplication, the source operands are always in the extended-precision floating-point format. If the source operands are in short or single-precision format, they are converted to extended-precision format. These conversions occur automatically in hardware with no overhead. All results of floating-point multiplications are returned in the extended-precision format.

A multiplication occurs in a single cycle.

Figure 5–15 is a flowchart showing the steps involved in a floating-point multiplication. Each step is labelled with a number in parentheses.

- In step 1, the 32-bit source mantissas, α(man) and b(man), are multiplied, producing a 64-bit result, c(man). (Note that input and output data are always represented as normalized numbers.)
- \Box In step 2, the exponents, $\alpha(exp)$ and b(exp), are added, yielding c(exp).
- Step 3 checks whether or not c(man) is equal to zero. If c(man) is zero, step 7 sets c(exp) to −128, thus yielding the representation for zero.
- Steps 4 and 5 normalize the result.
- ☐ If a right shift of one is necessary, then in step 8, c(*man*) is right-shifted one bit, and 1 is added to c(*exp*).
- ☐ If a right shift of two is necessary, then in step 9, c(*man*) is right-shifted two bits, and 2 is added to c(exp). step 6 occurs when the result is normalized.
- In step 10, c(*man*) is set in the extended-precision floating-point format.
- Steps 11 through 16 check for special cases of c(*exp*).
- In step 14, if c(*exp*) has overflowed (detected in step 11) in the positive direction, then c(*exp*) is set to the most positive extended-precision format value. If c(*exp*) has overflowed in the negative direction, then c(*exp*) is set to the most negative extended-precision format value.

□ If c(exp) has underflowed (detected in step 12), then c is set to zero in step 15; i.e., c(man) = 0 and c(exp) = -128.

Figure 5–15. Flowchart for Floating-Point Multiplication



Example 5–6 through Example 5–9 illustrate how floating-point multiplication is performed on the 'C4x. For these examples, the implied most significant nonsign bit is made explicit.

Example 5–6. Floating-Point Multiply (Both Mantissas = -2.0)

Let

 $\begin{aligned} \alpha &= -2.0 \times 2^{\alpha(exp)} = 10.0000000000000000000000 \times 2^{\alpha(exp)} \\ b &= -2.0 \times 2^{b(exp)} = 10.0000000000000000000 \times 2^{b(exp)} \\ \end{aligned}$ where a and b are both represented in binary form according to the normalized single-precision floating-point format.

To place this number in the proper normalized format, it is necessary to shift the mantissa two places to the right and add 2 to the exponent. This yields

```
\begin{array}{l} 10.0000000000000000000 \times 2^{\alpha(\textit{exp})} \\ \times \ 10.00000000000000000000 \times 2^{b(\textit{exp})} \end{array}
```

In floating-point multiplication, the exponent of the result may overflow when the exponents are initially added or when the exponent is modified during normalization.

Example 5–7. Floating-Point Multiply (Both Mantissas = 1.5)

Let

$\alpha = 1.5 \times 2^{\alpha(exp)}$	=	01.100000000000000000000000000000000000	$\times 2^{\alpha}(exp)$
b = 1.5 × 2 ^{b(<i>exp</i>)}	=	01.100000000000000000000000000000000000	$\times 2^{b(exp)}$

where α and b are both represented in binary form according to the single-precision floating-point format. Then

 $\begin{array}{rcrcrc} 01.100000000000000000 & \times & 2^{\alpha(\textit{exp})} \\ \times & 01.10000000000000000000 & \times & 2^{b(\textit{exp})} \end{array}$

To place this number in the proper normalized format, it is necessary to shift the mantissa one place to the right and add 1 to the exponent. This yields

 $\begin{array}{l} 01.1000000000000000000 \times 2 \alpha (\textit{exp}) \\ \times 01.100000000000000000000 \times 2 b (\textit{exp}) \end{array}$

Data Formats and Floating-Point Operation 5-21

Example 5–8. Floating-Point Multiply (Both Mantissas = 1.0)

Let $\alpha = 1.0 \times 2^{\alpha(exp)} = 01.000000000000000000 \times 2^{\alpha(exp)}$ $b = 1.0 \times 2^{b(exp)} = 01.00000000000000000 \times 2^{b(exp)}$

where a and b are both represented in binary form according to the single-precision floating-point format. Then

This number is in the proper normalized format. Therefore, no shift of the mantissa or modification of the exponent is necessary.

The previous three examples show cases in which the product of two normalized numbers can be normalized with a shift of zero, one, or two. The floating-point format of the 'C4x makes this possible.

Example 5–9. Floating-Point Multiply Between Positive and Negative Numbers

 $\begin{array}{l} \mbox{Let} & \\ \alpha = 1.0 \times 2^{\alpha(exp)} = 01.0000000000000000000 \times 2^{\alpha(exp)} \\ b = -2.0 \times 2^{b(exp)} = 10.0000000000000000 \times 2^{b(exp)} \\ \mbox{Then} & \\ 01.000000000000000000 \times 2^{\alpha(exp)} \\ \times 10.000000000000000 \times 2^{b(exp)} \end{array}$

The result is $c = -2.0 \times 2(\alpha(exp) + b(exp))$

Floating-Point Multiply by Zero

All multiplications by a floating-point zero yield a result of zero (f = 0, s = 0, and exp = -128).

5.6 Floating-Point Addition and Subtraction

In floating-point addition and subtraction, two floating-point numbers $\boldsymbol{\alpha}$ and b can be defined as

 $\alpha = \alpha(man) \times 2 \alpha(exp)$ b = b(man) × 2 b(exp)

The sum (or difference) of α and b can be defined as

 $\begin{aligned} \mathbf{c} &= \alpha \pm \mathbf{b} \\ &= (\alpha(man) \pm (\mathbf{b}(man) \times 2^{-(\alpha(exp) - \mathbf{b}(exp))})) \times 2^{-\alpha(exp)}, \\ &\text{if } \alpha(exp) \geq \mathbf{b}(exp) \\ &= ((\alpha(man) \times 2^{-(\mathbf{b}(exp) - \alpha(exp))}) \pm \mathbf{b}(man)) \times 2^{-\mathbf{b}(exp)}, \\ &\text{if } \alpha(exp) < \mathbf{b}(exp) \end{aligned}$

Figure 5–16 is the flowchart for floating-point addition. Because this flowchart assumes signed data, it is also appropriate for floating-point subtraction. In this figure, it is assumed that $\alpha(exp) \leq b(exp)$. Steps are shown as numbers in parentheses in the figure.

- In step 1, the source exponents are compared, and c(exp) is set equal to the largest of the two source exponents.
- In step 2, *d* is set to the difference of the two exponents.
- In step 3, the mantissa with the smallest exponent, in this case $\alpha(man)$, is right-shifted *d* bits in order to align the mantissas.
- In step 4, after the mantissas have been aligned, they are added.
- □ In steps 5 through 7 check for a special case of c(man). If c(man) is zero (step 5), then c(exp) is set to its most negative value (step 8) to yield the correct representation of zero. If c(man) has overflowed c (step 6), then in step 9, c(man) is right-shifted one bit, and 1 is added to c(exp). In step 10, the result is normalized.
- In steps 11 and 12, special cases of c(*exp*) are tested. If c(*exp*) has overflowed, then c is set to the most positive extended-precision value if it is positive; if it is negative, it is set to the most negative extended-precision value.



Figure 5–16. Flowchart for Floating-Point Addition

The following examples describe the floating-point addition and subtraction operations. It is assumed that the data is in the extended-precision floating-point format.

Example 5–10. Floating-Point Addition

Let

It is necessary to shift b to the right by one so that α and b have the same exponent. This yields

Then

As in the case of multiplication, it is necessary to shift the binary point one place to the left and to add 1 to the exponent. This yields

Example 5–11. Floating-Point Subtraction

Let

The operation to be performed is α -b. The mantissas are already aligned because the two numbers have the same exponent. The result is a large cancellation of the upper bits, as shown below.

Data Formats and Floating-Point Operation 5-25

The result must be normalized. In this case, a left shift of 31 is required. The exponent of the result is modified accordingly. The result is

Example 5–12. Floating-Point Addition With a 32-Bit Shift

This example illustrates a situation in which a full 32-bit shift is necessary to normalize the result. Let

The operation to be performed is α + b.

Normalizing the result requires a left shift of 32 and a subtraction of 32 from the exponent. The result is

Example 5–13. Floating-Point Addition/Subtraction and Zero

When floating-point addition and subtraction are performed with a floating-point 0, the following identities are satisfied:

 $\alpha \pm 0 = \alpha$ (assuming that $\alpha \neq 0$)

 $0\pm 0=0$

 $0 - \alpha = -\alpha$ (assuming that $\alpha \neq 0$)

5.7 Normalization (NORM Instruction)

The NORM instruction normalizes an extended-precision floating-point number that is assumed to be unnormalized. Since the number is assumed to be unnormalized, no implied most significant nonsign bit is assumed. The NORM instruction executes three steps:

- 1) Locates the most significant nonsign bit of the floating-point number
- 2) Left shifts to normalize the number
- 3) Adjusts the exponent

Given the extended-precision floating-point value α to be normalized, the normalization is performed as shown in Figure 5–17.

Figure 5–17. Flowchart for NORM Instruction Operation



Data Formats and Floating-Point Operation 5-27

Example 5–14. NORM Instruction

Assume that an extended-precision register contains the value:

When the normalization is performed on a number assumed to be unnormalized, the binary point is assumed to be:

This number is then sign extended one bit so that the mantissa contains 33 bits:

Here is the intermediate result after the most significant nonsign bit is located and the shift is performed:

The final 32-bit value output after removing the redundant bit is:

man = 00000000000100000000000000000000, *exp* = -19

The NORM instruction is useful for counting the number of leading zeros or leading ones in a 32-bit field. If the exponent is initially zero, the absolute value of the final value of the exponent is the number of leading ones or zeros. This instruction is also useful for manipulating unnormalized floating-point numbers.

5.8 Rounding (RND Instruction)

The RND instruction rounds a number from the extended-precision floating-point format to the single-precision floating-point format in a single cycle. Rounding (rnd) is similar to floating-point addition. Given the number α to be rounded, the following operation is performed first.

 $c = \alpha(man) \times 2^{\alpha(exp)} + (1 \times 2^{\alpha(exp)-24})$

Next, a conversion from extended-precision floating-point to single-precision floating-point format is performed. Given the extended-precision floating-point value, rounding is performed as shown in Figure 5–18.

Note:

RND *src*, *dst* — where (*src*) = 0 — does not set the zero condition flag (bit 2 in the status register). Instead, it sets the underflow condition flag (bit 4 in the status register). When required, check for the underflow condition instead of the zero condition.





5.9 Floating-Point-to-Integer Conversion (FIX Instruction)

Using the FIX instruction, you can convert an extended-precision floating-point number to a single-precision integer in a single cycle. The floating-point to integer conversion of the value x is referred to here as fix(x). The conversion does not overflow if α , the number to be converted, is in the range

 $-2^{31}\leq\alpha\leq2^{31}-1$

First, you must be certain that:

 $\alpha(exp) \leq 30$

If these bounds are not met, an overflow occurs. If an overflow occurs in the positive direction, the output is the most positive integer. If an overflow occurs in the negative direction, the output is the most negative integer. If $\alpha(exp)$ is within the valid range, then $\alpha(man)$, with implied bit included, is sign-extended and right-shifted (rs) by the amount:

 $rs = 31 - \alpha(exp)$

This right shift (rs) shifts out those bits corresponding to the fractional part of the mantissa. For example:

If $0 \le x < 1$, then fix(x) = 0. If $-1 \le x < 0$, then fix(x) = -1.

The flowchart for the floating-point-to-integer conversion is shown in Figure 5–19.



Figure 5–19. Flowchart for Floating-Point-to-Integer Conversion by FIX Instruction

5.10 Integer-to-Floating-Point Conversion (FLOAT Instruction)

Integer-to-floating-point conversion performed by the FLOAT instruction allows a single-precision integer to be converted to an extended-precision floating-point number in a single cycle. The flowchart for this conversion is shown in Figure 5–20.





5.11 Reciprocal (RCPF Instruction)

The RCPF instruction generates a satisfactory estimate of the reciprocal of a floating-point number in a single cycle. The estimate has the correct exponent, and the mantissa is accurate to the eighth binary position (mantissa error is thus $< 2^{-8}$) giving a 16-bit representation of the result (8-bit exponent plus 8-bit mantissa). Also, this estimate can be used as a seed for an algorithm to compute the reciprocal to even greater accuracy. (The Newton-Raphson algorithm, described in this section, is one such case.)

Figure 5–21 below depicts the algorithm used by instruction RCPF.

- The input is assumed to be $v = vman \times 2^{vexp}$.
- The output is assumed to be $x = xman \times 2^{xexp}$.
- vexp is negated.
- If vexp = -128, the result is saturated to the most positive number, and the overflow flag is set. The N condition flag is set to the same sign as *vsign*.

Figure 5–21. RCPF Instruction Algorithm



The look-up table is read by forming a nine-bit address consisting of *vsign* and bits 22–15 of *vfrac*. The eight-bit output of the look-up table forms bits 22–15 of *xfrac*. Bits 14–0 of *xfrac* are cleared to zero. *xsign* is set to *vsign*.

The look-up table values are generated from simulation results.

5.11.1 Reciprocal Algorithm

The RCPF instruction provides the reciprocal of a number. The estimate has the correct exponent and a mantissa accurate to the eighth binary place (i.e., the error of the mantissa is $< 2^{-8}$). The Newton-Raphson algorithm (shown below) can be used to further extend the mantissa's precision:

$$x[n+1] = x[n](2 - vx[n])$$

where v = the number whose reciprocal is to be found.

x[0], the seed for the algorithm, is given by RCPF. For each iteration of the algorithm, the number of accurate bits in the mantissa doubles. Using RCPF, you can start with an estimate accurate to eight bits. With one iteration, accuracy is 16 bits in the mantissa, and with a second iteration, accuracy is 32 bits.

The 'C4x program to implement this algorithm is shown in Example 5–15. Each step of the algorithm is labeled along with the corresponding accuracy achieved at the end of the step. The algorithm takes only seven machine cycles.

Example 5–15. Newton-Raphson Algorithm for Computing the Reciprocal

```
R0, R1; R0 = v, R1 = x[0]
   RCPF
   MPYF
          R1, R0, R2
   SUBRF
         2.0,R2
   MPYF
         R2,R1 ; end of first iteration (16-bit accuracy)
          R1, R0, R2
   MPYF
   SUBRF 2.0,R2
   MPYF
         R2,R1 ; end of second iteration (32-bit accuracy)
;
                ; R1 = 1/v
;
;
```

5.12 Reciprocal Square Root (RSQRF Instruction)

In many applications, normalization of data values is necessary. Often, the normalizing factor is the square root of another quantity. For example, when one vector is given, you can find the unit vector in the same direction by dividing the original vector by its own length. This involves division by a square root. The RSQRF instruction provides a simple way to directly determine this quantity instead of going through a two-step approach of finding the square root and then finding the reciprocal of the square root.

Given the result of this algorithm, the square root is found by a simple multiplication:

v = vx[n]

where x[n] is the estimate of $\frac{1}{\sqrt{v}}$ as determined by the Newton-Raphson algorithm or some other algorithm.

The RSQRF instruction generates an estimated reciprocal of the square root of a floating-point number in a single cycle. It parallels some of the operational characteristics of the RCPF instruction in these ways:

- RSQRF generates an estimate (in this case, the reciprocal of the square root of a floating-point number).
- The mantissa is accurate to the eighth binary place (mantissa error is $< 2^{-8}$).
- Often, this is a satisfactory estimate of the reciprocal of a number's square root; in other cases, it may be used as a seed for an algorithm that computes the reciprocal square root to an even greater accuracy.

Figure 5–22 depicts the RSQRF algorithm. In the algorithm:

- The input is assumed to be $v = vman \times 2^{vexp}$.
- The output is assumed to be $x = xman \times 2^{xexp}$.
- \Box vexp + 1 is negated and shifted right one bit with sign extension.
- If vexp = -128, the result is saturated to the most positive number, and the overflow flag is set.





The look-up table is read by forming a nine-bit address consisting of the least significant bit of *vexp* and bits 22-15 of *vfrac*. The eight-bit output of the look-up table forms bits 22-15 of *xfrac*. Bits 14-0 of *xfrac* are cleared to zero. *xsign* is set to 0. There is no provision for negative values of *v*.

The look-up table values are generated from simulation results.

Given the result of this algorithm, division is performed by a simple multiplication:

y/v = yx[n]

In the equation, x[n] is the estimate of 1/v as determined by the Newton-Raphson algorithm or another algorithm.

Newton-Raphson Algorithm

The RSQRF instruction provides the reciprocal of the square root of a number. The estimate has the correct exponent and a mantissa accurate to the eighth binary place (i.e., the error of the mantissa is $< 2^{-8}$). The Newton-Raphson algorithm (shown below) can be used to further extend the mantissa's precision:

x[n+1] = x[n](1.5 - (v/2)x[n]x[n])

where v = the number whose reciprocal is to be found.

Data Formats and Floating-Point Operation 5-37

The seed for the algorithm, x[0], is given by RSQRF. For each iteration of the algorithm, the number of accurate bits in the mantissa doubles. Using RSQRF, you can start with an estimate accurate to eight bits. With one iteration, accuracy is 16 bits in the mantissa, and with a second iteration, accuracy is 32 bits.

The 'C4x program to implement this algorithm is shown in Example 5–16. Each step of the algorithm is labeled, and the corresponding accuracy achieved is noted at the end of the step. The algorithm takes only ten machine cycles (compared to 30 cycles on the 'C3x without a look-up table).

Example 5–16. Newton-Raphson Algorithm for Computing the Reciprocal Square Root

```
RSQRF R0,R1 ; R0 = v, R1 = x[0]
   MPYF
         0.5, R0; R0 = v/2
;
         R1,R1,R2
   MPYF
   MPYF R0,R2
   SUBRF 1.5,R2
   MPYF R2,R1 ; end of first iteration (16-bit accuracy)
;
   MPYF
         R1,R1,R2
   MPYF
         R0,R2
   SUBRF 1.5,R2
   MPYF R2,R1 ; end of second iteration (32-bit accuracy)
;
                ; R1 = 1/(v^{*}0.5)
;
;
```

Chapter 6

Addressing Modes

The 'C4x supports five types of addressing to access data from memory, registers, and the instruction word. This chapter details the operation, encoding, and implementation of the addressing modes.

Topic

Page

6.1	Addressing Types 6-2
6.2	Register Addressing 6-3
6.3	Direct Addressing
6.4	Indirect Addressing 6-6
6.5	Immediate Addressing 6-18
6.6	PC-Relative Addressing
6.7	Encoding of Addressing Modes
6.8	Circular Addressing
6.9	Bit-Reversed Addressing

6.1 Addressing Types

You can access data from memory, registers, and the instruction word by using five types of addressing:

- Register addressing
- Direct addressing
- Indirect addressing
- □ Immediate addressing
- PC-relative addressing

Not all addressing types are appropriate for all instructions. Addressing types are classified into four groups, depending upon the encoding method used:

- General addressing modes (G)
- □ Three-operand addressing modes (T)
- Parallel addressing modes (P)
- Conditional-branch addressing modes (B)

For use in filters and FFTs, there are two specialized modes:

- Circular addressing
- Bit-reversed addressing

6.2 Register Addressing

In register addressing, a CPU register contains the operand, as shown in this example:

ABSF R1 ; R1 = |R1|

The machine address for the CPU registers, the assembler syntax (register name), and the assigned function for those registers are listed in Table 6–1.

Table 6–1. CPU Register/Assembler Syntax and Function

(a) CPU Primary Registers			
Register Name	Machine Address	Assigned Function	
R0	00h	Extended-precision register 0	
R1	01h	Extended-precision register 1	
R2	02h	Extended-precision register 2	
R3	03h	Extended-precision register 3	
R4	04h	Extended-precision register 4	
R5	05h	Extended-precision register 5	
R6	06h	Extended-precision register 6	
R7	07h	Extended-precision register 7	
R8	1Ch	Extended-precision register 8	
R9	1Dh	Extended-precision register 9	
R10	1Eh	Extended-precision register 10	
R11	1Fh	Extended-precision register 11	
A0	08h	Auxiliary register 0	
A1	09h	Auxiliary register 1	
A2	0Ah	Auxiliary register 2	
A3	0Bh	Auxiliary register 3	
A4	0Ch	Auxiliary register 4	
A5	0Dh	Auxiliary register 5	
A6	0Eh	Auxiliary register 6	

Register Addressing

Table 6–1. CPU Register/Assembler Syntax and Function (Continued)

Register Name	Machine Address	Assigned Function
A7	0Fh	Auxiliary register 7
DP	10h	Data-page pointer
IR0	11h	Index register 0
IR1	12h	Index register 1
BK	13h	Block-size register
SP	14h	Active stack pointer
ST	15h	Status register
DIE	16h	DMA coprocessor interrupt enable
IIE	17h	Internal interrupt enable register
lif	18h	IIOF pins and interrupt flag register
RS	19h	Repeat start address register
RE	1Ah	Repeat end address register
RC	1Bh	Repeat counter register

(b) CPU Expansion Registers

Register Name	Machine Address	Assigned Function
IVTP	00h	Interrupt-vector table pointer
TVTP	01h	Trap-vector table pointer

6.3 Direct Addressing

In direct addressing, the data address is formed by the concatenation of the 16 least significant bits of the data page pointer (DP) with the 16 least significant bits of the instruction word (expr). The use of 16 bits for the DP results in 65536 pages (64K words per page), allowing you to access a large address space without changing the value of the DP. The syntax and operation for direct addressing are listed below.

Syntax: @expr

Operation: address = DP concatenated with expr

Figure 6–1 shows the formation of the data address. Example 6–1 gives an instruction example with data before and after instruction execution.

Figure 6–1. Direct Addressing





ADDI @OBCDEh,R7	
Before Instruction:	After Instruction:
DP = 108Ah	DP = 108Ah
R7 = 11h	R7 = 1234 5689h
Data at 108A BCDEh = 1234 5678h	Data at 108A BCDEh = 1234 5678h

6.4 Indirect Addressing

Indirect addressing specifies the address of an operand in memory through the contents of an auxiliary register, optional displacements, and index registers. The auxiliary register arithmetic units (ARAUs) perform this unsigned arithmetic. (All 32 bits of the auxiliary and index registers are used in indirect addressing.)

The flexibility of indirect addressing is possible because the ARAUs on the 'C4x modify auxiliary registers in parallel with operations within the main CPU. Indirect addressing is specified by a five-bit field in the instruction word, referred to as the mod field (shown on the left side of Table 6–2 on as well as in the examples that follow). A displacement is either an explicit unsigned 5-bit or 8-bit integer contained in the instruction word or an implicit displacement of one. Two index registers, IR0 and IR1, can also be used in indirect addressing, enabling the use of 32-bit indirect displacements (IR0 and IR1 are treated as signed integers). In some cases, an addressing scheme using circular or bit-reversed addressing is optional. Generating addresses for circular addressing is discussed in Section 6.8, and for bit-reversed addressing in Section 6.9.

Table 6–2 lists the various kinds of indirect addressing, along with the value of the modification (mod) field, assembler syntax, operation, and function for each. Figure 6–2 shows the format of the indirect addressing operand in the instruction encoding. The *disp* field does not exist for some instructions.

Figure 6–2. Indirect Addressing Operand Encoding

MSB		LSB
mod	ARn	disp
5 bits	3 bits	0, 5, or 8 bits

Note:

The auxiliary register (AR*n*) to be used is encoded in the instruction word according to its binary representation, n (i.e., AR3 is encoded as 11₂), not its register machine address (as shown in Table 6–1).

Table 6–2. Indirect Addressing

(a) Indirect Addressing With Displacement

Mod Field	Syntax	Operation	Description
00000	*+ARn(disp)	addr = ARn + disp	With predisplacement add
00001	*–ARn(disp)	addr = ARn - disp	With predisplacement subtract
00010	*++ARn(disp)	addr = ARn + disp ARn = ARn + disp	With predisplacement add and modify
00011	*ARn(disp)	addr = ARn – disp ARn = ARn – disp	With predisplacement subtract and modify
00100	*ARn++(disp)	addr = ARn ARn = ARn + disp	With postdisplacement add and modify
00101	*ARn(disp)	addr = ARn ARn = ARn – disp	With postdisplacement subtract and modify
00110	*ARn++(disp)%	addr = ARn ARn = circ(ARn + disp)	With postdisplacement add and circular modify
00111	*ARn(disp)%	add = ARn ARn = circ(ARn – disp)	With postdisplacement subtract and circular modify

(b) Indirect Addressing With Index Register IRO

Mod Field	Syntax	Operation	Description
01000	*+ARn(IR0)	addr = ARn + IR0	With preindex (IR0) add
01001	*-ARn(IR0)	addr = ARn - IR0	With preindex (IR0) subtract
01010	*++ARn(IR0)	addr = ARn + IR0 ARn = ARn + IR0	With preindex (IR0) add and modify
01011	*ARn(IR0)	addr = ARn – IR0 ARn = ARn – IR0	With preindex (IR0) subtract and modify
01100	*ARn++(IR0)	addr = ARn ARn = ARn + IR0	With postindex (IR0) add and modify
01101	*ARn(IR0)	addr= ARn ARn = ARn – IR0	With postindex (IR0) subtract and modify
01110	*ARn++(IR0)%	addr = ARn ARn = circ(ARn + IR0)	With postindex (IR0) add and circular modify
01111	*ARn(IR0)%	addr = ARn ARn = circ(ARn- IR0)	With postindex (IR0) subtract and circular modify
LEGEND: addr = ARn = IRn =	memory address auxiliary register AF index register IR0 o	R0 – AR7 circ() = r IR1 % =	subtract and modify address in circular addressing where circular addressing is performed

++ = add and modify

= displacement

disp

Addressing Modes 6-7

Mod Field	Syntax	Operation	Description
10000	*+ARn(IR1)	addr = ARn + IR1	With preindex (IR1) add
10001	*-ARn(IR1)	addr = ARn - IR1	With preindex (IR1) subtract
10010	*++ARn(IR1)	addr = ARn + IR1 ARn = ARn + IR1	With preindex (IR1) add and modify
10011	*ARn(IR1)	addr = ARn – IR1 ARn = ARn – IR1	With preindex (IR1) subtract and modify
10100	*ARn++(IR1)	addr = ARn ARn = ARn + IR1	With postindex (IR1) add and modify
10101	*ARn(IR1)	addr = ARn ARn = ARn – IR1	With postindex (IR1) subtract and modify
10110	*ARn++(IR1)%	addr = ARn ARn = circ(ARn + IR1)	With postindex (IR1) add and circular modify
10111	*ARn(IR1)%	addr = ARn ARn = circ(ARn – IR1)	With postindex (IR1) subtract and circular modify

Table 6–2. Indirect Addressing (Continued)

(c) Indirect Addressing With Index Register IR1

(d) Indirect Addressing (Special Cases)

Mod Field	Syntax	Operation	Description
11000	*ARn	addr = ARn	Indirect
11001	*ARn++(IR0)B	addr = ARn ARn = B(ARn + IR0)	With postindex (IR0) add and bit-reversed modify
LEGEND: addr ARn IRn disp ++	 memory address auxiliary register AR index register IR0 or displacement add and modify 	0 – AR7 circ() IR1 % B	 subtract and modify address in circular addressing where circular addressing is performed where bit-reversed addressing is performed

Example 6–2 through Example 6–19 show the operation for each type of indirect addressing.

Example 6–2. Auxiliary Register Indirect

An auxiliary register (ARn) contains the address of the operand to be fetched.



Example 6-3. Indirect With Predisplacement Add

The address of the operand to be fetched is the sum of an auxiliary register (ARn) and the displacement (*disp*). The displacement is either a 5-bit or 8-bit unsigned integer contained in the instruction word or an implied value of 1.



Example 6-4. Indirect With Predisplacement Subtract

The address of the operand to be fetched is the contents of an auxiliary register (ARn) minus the displacement (disp). The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1.



Example 6-5. Indirect With Predisplacement Add and Modify

The address of the operand to be fetched is the sum of an auxiliary register (AR*n*) and the displacement (*disp*). The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1. After the data is fetched, the auxiliary register is updated with the generated address.



Example 6-6. Indirect With Predisplacement Subtract and Modify

The address of the operand to be fetched is the contents of an auxiliary register (AR*n*) minus the displacement (*disp*). The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1. After the data is fetched, the auxiliary register is updated with the generated address.



Example 6–7. Indirect With Postdisplacement Add and Modify

The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the displacement (*disp*) is added to the auxiliary register. The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1.

Operation:	operand address = AR <i>n</i> AR <i>n</i> = AR <i>n</i> + <i>disp</i>
Assembler Syntax:	*AR <i>n</i> ++ disp
Modification Field:	00100
31	



Addressing Modes 6-11
Example 6–8. Indirect With Postdisplacement Subtract and Modify

The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the displacement (*disp*) is subtracted from the auxiliary register. The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1.



Example 6–9. Indirect With Postdisplacement Add and Circular Modify

The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the displacement (*disp*) is added to the contents of the auxiliary register through circular addressing. This result is used to update the auxiliary register. The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1.

Operation: op AF Assembler Syntax: *A Modification Field: 00

operand address = ARn ARn = circ(ARn + disp) *ARn ++ (disp)% 00110



Example 6–10. Indirect With Postdisplacement Subtract and Circular Modify

The address of the operand to be fetched is the contents of an auxiliary register (AR*n*). After the operand is fetched, the displacement (*disp*) is subtracted from the contents of the auxiliary register through circular addressing. This result is used to update the auxiliary register. The displacement is either an 8-bit unsigned integer contained in the instruction word or an implied value of 1.





The address of the operand to be fetched is the sum of an auxiliary register (ARn) and an index register (IR0 or IR1).



Example 6–12. Indirect With Preindex Subtract

The address of the operand to be fetched is the difference between an auxiliary register (AR*n*) and an index register (IR0 or IR1).



Example 6–13. Indirect With Preindex Add and Modify

The address of the operand to be fetched is the sum of an auxiliary register (AR*n*) and an index register (IR0 or IR1). After the data is fetched, the auxiliary register is updated with the generated address.

Operation:		operand address = AR <i>n</i> +IR <i>m</i> AR <i>n</i> = AR <i>n</i> +IR <i>m</i>		
Assembler syntax:		*++AR <i>n</i> (IR <i>m</i>)		
Modification Field:		01010 10010	if m=0 if m=1	
_31				0
ARn —		Address		
31			0	
IRm →	Index		(+)	
_31			\	0
		Operand		

Example 6–14. Indirect With Preindex Subtract and Modify

The address of the operand to be fetched is the difference between an auxiliary register (AR*n*) and an index register (IR0 or IR1). The resulting address becomes the new contents of the auxiliary register.

Operation:		operand address = AR <i>n</i> -IR <i>m</i> AR <i>n</i> = AR <i>n</i> -IR <i>m</i>		
Assembler Syntax:		*——AR <i>n</i> (IR <i>m</i>)		
Modification Field:		01011 10011	if m=0 if m=1	
	31			0
ARn —		Address		
31			0	^
IRm	Index			
	31			0
[Operand		

Example 6–15. Indirect With Postindex Add and Modify

The address of the operand to be fetched is the contents of an auxiliary register (AR*n*). After the operand is fetched, an index register (IR0 or IR1) is added to the auxiliary register.

Operation:		operand address AR <i>n</i> = AR <i>n</i> +IR <i>r</i>	s = AR <i>n</i> n
Assembler Syntax	:	*AR <i>n</i> ++(IR <i>m</i>)	
Modification Field	:	01100 10100	if m=0 if m=1
	_31		0
ARn	-	Address	
31			<u> </u>
IRm →	Index		(+) (+)
	31		<u> </u>
		Operand	

Example 6–16. Indirect With Postindex Subtract and Modify

The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the index register (IR0 or IR1) is sub-tracted from the auxiliary register.

Operation:	operand address = . AR <i>n</i> = AR <i>n</i> -IR <i>m</i>	AR <i>n</i>
Assembler Syntax:	*AR <i>n</i> (IR <i>m</i>)	
Modification Field:	01101 10101	if m = 0 if m = 1



Example 6–17. Indirect With Postindex Add and Circular Modify

The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the index register (IR0 or IR1) is added to the auxiliary register. This value is evaluated through circular addressing and replaces the contents of the auxiliary register.

Operation:	operand address = AR <i>n</i> AR <i>n</i> = circ(AR <i>n</i> + IR <i>m</i>)		
Assembler Syntax:	*AR <i>n</i> ++(IR <i>m</i>))%	
Modification Field:	01110	if m=0	
	10110	if m=1	





The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the index register (IR0 or IR1) is sub-tracted from the auxiliary register. The result is evaluated through circular addressing and replaces the contents of the auxiliary register.



Example 6–19. Indirect With Postindex Add and Bit-Reversed Modify

The address of the operand to be fetched is the contents of an auxiliary register (ARn). After the operand is fetched, the index register (IR0) is added to the auxiliary register. This addition is performed with a reverse-carry propagation and can be used to yield a bit-reversed (B) address. This value replaces the contents of the auxiliary register.



Addressing Modes 6-17

6.5 Immediate Addressing

In immediate addressing, the operand is an 8- or 16-bit immediate value contained in the 8 or 16 least significant bits of the instruction word (expr). Depending on the data types assumed for the instruction, the immediate operand may be a twos-complement integer, an unsigned integer, a signed integer, or a floating-point number. The syntax for this mode is as follows:

Syntax: expr

Example 6–20 gives an instruction example with data from before and after the instruction is executed. Notice that AND and AND3 produce different results.

Example 6–20. Immediate Addressing

Instruc	tion	Before			After		
SUBI	1,R0	R0=0h			R0=00	FFFF	FFFFh
LDI	OFFFFh,R0	R0=0h			R0=00	FFFF	FFFFh
LDF	5.0,R0	R0=0h			R0=02	2000	0000h
OR	OFFFFh,R0	R0=0h			R0=00	0000	FFFFh
AND 3	80h,R0,R0	R0=00	FFFF	FFFFh	R0=00	FFFF	FF80h
AND	80h,R0	R0=00	FFFF	FFFFh	R0=00	0000	0080h

6.6 PC-Relative Addressing

PC-relative addressing is used for branching. It adds the contents of the 16 or 24 least significant bits of the instruction word to the PC register. The assembler takes the *src* (a label or address) specified by the user and generates a displacement. If the branch is a standard branch, this displacement is equal to [*label* – (*instruction address*+1)]. If the branch is a delayed branch, this displacement is equal to [*label* – (*instruction address*+3)].

The displacement is stored as a 16-bit or 24-bit signed integer in the least significant bits of the instruction word. The displacement is added to the PC during the pipeline decode phase. Notice that because the PC is incremented by one in the fetch phase, the displacement is added to this incremented PC value.

Syntax: expr (label or address)

Example 6–21 gives an instruction example with before- and after-instruction data.

Example 6–21. PC-Relative Addressing

NEWPC	BU 	NEWPC	; ; ;	address of BU instruction=1, NEWPC label =5, displacement = 3 displacement = 5 - (1 + 1)
Before Decode	Instruc e Phase	tion		After Instruction Execution Phase:
PC = 2	h			PC = 5h

The 24-bit addressing mode is used to encode the program control instructions (e.g., BR, BRD, CALL, RPTB, RPTBD, LAJ). Depending on the instruction, the new PC value is derived by adding a 24-bit signed value in the instruction word with the present PC value. Bit 24 determines the type of branch (D = 0 for a standard branch or D = 1 for a delayed branch). Some of these instructions are encoded in Figure 6–3.

Figure 6–3. Encoding for 24-Bit PC-Relative Addressing Mode

				(a)	E	BR,	BRD: unconditional branches (delayed and not delayed)	
31					25	5	23	0
0	1	1	0	0 0	0	D	SIC	
				(b)	C	CAL	L: unconditional subroutine call	
31							23	0
0	1	1	0	0 0	1	0	SrC	
				(C)	F	۲P	FB, RPTBD: repeat block (not delayed and delayed)	
31				(c)	F	P٦	FB, RPTBD: repeat block (not delayed and delayed) 23	0
31 0	1	1	1	(c) 1 0	F 0	₹P D	TB, RPTBD: repeat block (not delayed and delayed) 23 src	0
31 0	1	1	1	(c) 1 0	F 0	₹P⊺ D	TB, RPTBD: repeat block (not delayed and delayed) 23 src	0
31 0	1	1	1	(c) 1 0 (d)	F 0 L	₹₽٦ D	TB, RPTBD: repeat block (not delayed and delayed) 23 src : link and jump (return address in extended-precision	0
31 0	1	1	1	(c) 1 0 (d)	F O L	{P⊺ D .AJ	TB, RPTBD: repeat block (not delayed and delayed) 23 src : link and jump (return address in extended-precision ster R11)	0
31 0 31	1	1	1	(c) 1 0 (d)	F 0 L	}P⊺ D .AJ egi	TB, RPTBD: repeat block (not delayed and delayed) 23 src : link and jump (return address in extended-precision ster R11) 23	0

6.7 Encoding of Addressing Modes

The five addressing types form four groups of addressing modes:

- General addressing modes (G) (subsection 6.7.1)
- Three-operand addressing modes (T) (subsection 6.7.2)
- Parallel addressing modes (P) (subsection 6.7.3)
- Conditional-branch addressing modes (B) (subsection 6.7.4)

6.7.1 General Addressing Modes

Instructions that use the general addressing modes are general-purpose instructions, such as ADDI, MPYF, and LSH. Such instructions usually have the following syntax:

dst operation $src \rightarrow dst$

In the syntax, the destination operand is signified by *dst* and the source operand by *src*; operation defines an operation to be performed with the general addressing modes to specify certain operands. Bits 31-29 are zero, indicating general addressing mode instructions. Bits 22 and 21 specify the general addressing mode (G) field, which defines how bits 15 through 0 are to be interpreted for addressing the *src* operand.

G	Mode
00	register (all CPU registers unless specified otherwise)
01	direct
10	indirect
11	immediate

Options for bits 22 and 21 (G field) are as follows:

If the *src* and *dst* fields contain register specifications, the value in these fields contains the CPU register addresses as defined by Table 6–1. For the general addressing modes, the following values of AR*n* are valid for indirect addressing:

 $ARn, 0 \le n \le 7$

Figure 6–4 shows the encoding for the general addressing modes. The notation mod*n* indicates the modification field that goes with the AR*n* field. Refer to Table 6–2 for further information.

				G	Destination		Source C	Operands	-
31		29	28 23	22 21	20 16	15 11	10 8	75	4 0
0	0	0	operation	0 0	dst	00000	000	000	00000
0	0	0	operation	0 1	dst		direct		
0	0	0	operation	1 0	dst	mod <i>n</i>	AR <i>n</i>		disp
0	0	0	operation	1 1	dst	Immediate			

Figure 6–4. Encoding for General Addressing Modes

6.7.2 Three-Operand Addressing Modes

The 19 three-operand instructions on the 'C4x use the eight addressing modes listed in Table 6–3:

Table 6–3. Three-O	perand	Instruction .	Addres	ssing	Mode	ЭS

Type 1[†]

т	<i>src</i> 1 addressing modes	src2 addressing modes	dst‡
00	Register mode (any CPU register)	Register mode (any CPU register)	Rx
01	Indirect mode (<i>disp</i> = 0, 1, IR0, IR1)	Register mode (any CPU register)	Rx
10	Register mode (any CPU register)	Indirect mode (<i>disp</i> = 0, 1, IR0, IR1)	Rx
11	Indirect mode (<i>disp</i> = 0, 1, IR0, IR1)	Indirect mode (<i>disp</i> = 0, 1, IR0, IR1)	Rx

[†] The 'C4x recognizes either type 1 or type 2 modes; the 'C3x recognizes only type 1. [‡] Rx = any register in the CPU (primary) register file for the respective processor.

Type 2[†]

Т	src1 addressing modes	<pre>src2 addressing modes</pre>	dst ‡
00	Register mode (any CPU register)	8-bit signed immediate	Rx
01	Register mode (any CPU register)	Indirect mode *+ARn(5-bit unsigned displacement)	Rx
10	Indirect mode *+ARn(5-bit unsigned displacement)	8-bit signed immediate	Rx
11	Indirect mode *+ARn1(5-bit unsigned displacement)	Indirect mode *+ARn2(5-bit unsigned displacement)	Rx

[†] The 'C4x recognizes either type 1 or type 2 modes; the 'C3x recognizes only type 1. [‡] Rx = any register in the CPU (primary) register file for the respective processor. The object values differ for three-operand instructions, depending on the assembler used:

- The 'C3x assembler recognizes only type 1 modes and sets bits 31–28 to 00102.
- ☐ The 'C4x assembler recognizes both types and sets bits 31–28 to 0010₂ for type 1 and to 0011₂ for type 2.

The three-operand instructions MPYSHI3 and MPYUHI3 are unique to the 'C4x.

All instructions except four can use all of the type 2 address modes shown in Table 6–3. The exceptions, which can use only the second and fourth address modes in type 2, are the floating-point instructions ADDF3, CMPF3, MPYF3, and SUBF3.

The remaining 15 three-operand instructions are ADDC3, ADDI3, AND3, ANDN3, ASH3, CMPI3, LSH3, MPYI3, MPYSHI3, MPYUHI3, OR3, SUBB3, SUBI3, TSTB3, and XOR3.

Note:

The suffix 3 can be omitted from a three-operand instruction mnemonic.

Bits 22 and 21 specify the three-operand addressing mode (T) field, which defines how to interpret bits 15-0 for addressing the *src* operands. Bits 15-8 define the *src1* address, and bits 7-0 define the *src2* address.

Figure 6–5 and Figure 6–6 show the encoding for 'C4x three-operand addressing (the 'C3x recognizes only the format in Figure 6–5). The notation *modm* or *modn* indicates the modification field that goes with the AR*m* or AR*n* (auxiliary register) field, respectively. Refer to Table 6–2 for further information.

The 8-bit signed immediate value supports left shifts, right shifts, and memory increment and decrement operations. The immediate value is not available for floating-point operations.

These instructions greatly help reduce code size, both assembled and compiled. They also improve performance notably in DSP and other computationally intensive applications and general-purpose code.

							Г	Destination	src1									src2			
3-			28	27 23	;	22	21	20 16	15	15 13 12 11 ⁻			10	8	7		5	4 3	3	2	0
0	0	1	0	operation		0	0	dst	0 0 0 src1			src1		0	0	0		src	:2		
0	0	1	0	operation		0	1	dst		mod <i>n</i>			AF	₹n	0	0	0		src	2	
0	0	1	0	operation		1	0	dst	0	0 0 <i>src1</i>						mod	n		AF	٦n	
0	0	1	0	operation		1	1	dst	mod <i>n</i> AR <i>n</i>			ln		1	mod	m		AF	₹ <i>m</i>		

Figure 6–5. Encoding for Type 1 Three-Operand Addressing Modes ('C3x and 'C4x)

Figure 6–6. Encoding for Type 2 Three-Operand Addressing Modes ('C4x Only)

						Т	Destina	tion				src1				src	2		
3	1		28	27 23	22	21	20	16	15		13	12 11	10 8	7	5	4	3	2	0
0	0	1	1	operation	0	0	dst		0	0	0	R	ln		In	nmed	iate		
0	0	1	1	operation	0	1	dst		0	0	0	F	Rn		dis	Ø		A	R <i>n</i>
0	0	1	1	operation	1	0	dst				dis	ס	AR <i>n</i>		in	nmed	iate		
0	0	1	1	operation	1	1	dst		disp			AR <i>n</i>		dis	ס		A	R <i>m</i>	

6.7.3 Parallel Addressing Modes

Instructions that use parallel addressing, indicated by || (two vertical bars), allow for the greatest amount of parallelism possible. The destination operands are indicated as d1 and d2, signifying *dst1* and *dst2*, respectively (see Figure 6–4). The source operands, signified by *src1* and *src2*, use the extended-precision registers. The parallel operation to be performed is called operation.

Figure 6–7. Encoding for Parallel Multiply With ADD/SUB

31	30	29 26	6 25 24	23	22	21 19	18 16	15 11	10 8	7 3	2 0
1	0	operation	Р	d1	d2	src1	src2	mod <i>n</i>	AR <i>n</i>	mod <i>m</i>	AR <i>m</i>

The parallel addressing mode (P) field specifies how to use the operands, i.e., whether they are source or destination. The specific relationship between the P field and the operands is detailed in the description of the individual parallel instructions (see Chapter 14 for more information). However, the operands are always encoded in the same way. Bits 31 and 30 are set to the value of 10, indicating parallel addressing mode instructions. Bits 25 and 24 specify the parallel addressing mode (P) field, which defines how bits 21 - 0 are to be interpreted for addressing the *src* operands. Bits 21 - 19 define the *src1* address, bits 18 - 16 define the *src2* address, bits 15 - 8 the *src3* address, and bits 7 - 0 the *src 4* address. The notations mod *n* and mod *m* indicate the modification

field that goes with the AR*n* or AR*m* (auxiliary register) field, respectively. The parallel addressing operands are listed below.

src1 = Rn	$(0 \le n \le 7 \text{ for extended-precision registers } R0-R7)$
src2 = Rn	$(0 \le n \le 7 \text{ for extended-precision registers } R0-R7)$
d1	If 0, <i>dst1</i> is R0. If 1, <i>dst1</i> is R1.
d2	If 0, <i>dst2</i> is R2. If 1, <i>dst2</i> is R3.
Р	$0 \le P \le 3$
src3	indirect (<i>disp</i> = 0, 1, IR0, IR1)
src4	indirect (<i>disp</i> = 0, 1, IR0, IR1)

Note:

Only registers R0–R7 are used in parallel instructions. R8–R11 are not used in parallel instructions.

As in the three-operand addressing mode, indirect addressing in the parallel addressing mode allows for displacements of 0 or 1 and the use of the index registers (IR0 and IR1). The displacement of 1 is implied and is not explicitly coded in the instruction word.

In the encoding shown for this mode in Figure 6–7, if the *src3* and *src4* fields use the same auxiliary register, both addresses are correctly generated, but only the value created by the *src3* field is saved in the specified auxiliary register. The assembler issues a warning if you specify the same auxiliary register *src3* and *src4*.

6.7.4 Conditional-Branch Addressing Modes

Instructions using the conditional-branch addressing modes (B*cond*, B*cond*D, CALL*cond*, DB*cond*, and DB*cond*D) can perform a variety of conditional operations. Bits 31–27 are set to the value of 01101, indicating conditional-branch addressing mode instructions. Bit 26 is set to 0 or 1; 0 selects DB*cond*, and 1 selects B*cond*. Bit 25 determines the conditional-branch addressing mode (B). If B = 0, register addressing is used; if B = 1, PC-relative addressing is used. Bit 21 sets the type of branch: D = 0 for a standard branch, and D = 1 for a delayed branch. The condition field(*cond*) specifies the condition checked to determine what action to take — for example, whether or not to branch (see Table 14–8 on page 14-14 for a list of condition codes). Figure 6–6 shows the encoding for conditional-branch addressing.

Encoding of Addressing Modes

Figure 6–8. Encoding for Conditional-Branch Addressing Modes

DBcond (D):

3		26	25	24 22	21	20 16	15 5 4	0					
0	1 1 0 1	1	В	AR <i>n</i>	D	cond	0 0 0 0 0 0 0 0 0 0 0 0 0 <i>src</i> reg						
0	1 1 0 1	1	В	AR <i>n</i>	D	cond	Immediate (PC relative)						

Bcond (D):

31					26	25	24		22	21	20	16	15										5	4		0
0	1	1	0	1	0	В	0	0	0	D		cond	0	0	0	0	0	0	0	0	0	0	0		<i>src</i> reg	
0	1	1	0	1	0	В	0	0	0	D		cond	Immediate (PC relative)													

CALLcond:

31					26	25	24		22	21	20	16	15	5									5	4		0
0	1	1	1	0	0	В	0	0	0	D		cond	0	0	0	0	0	0	0	0	0	0	0		<i>src</i> reg	
0	1	1	1	0	0	В	0	0	0	D		cond	Immediate (PC relative)													

6.8 Circular Addressing

Many DSP algorithms require a circular buffer in memory. In convolution and correlation, a circular buffer acts as a sliding window that contains the most recent data to be processed. As new data is brought in, the new data overwrites the oldest data. The key to using a circular buffer is the implementation of a circular addressing mode. This section describes the circular addressing mode of the 'C4x.

The block-size register (BK) specifies the size of the circular buffer. If the most significant bit equal to 1 in the BK register is labeled bit N, with $N \le 15$, the address immediately following the bottom of the circular buffer can be found by concatenating bits 31 through N+1 of a user-selected register (AR*n*) with bits N through 0 of the BK register. The address of the top of the buffer is referred to as the effective base (EB) and can be found by concatenating bits 31 through 0 of EB are zero.

Figure 6–9 illustrates the relationships among the block-size register (BK), the auxiliary registers (ARn), the bottom of the circular buffer, the top of the circular buffer, and the index into the circular buffer.

A circular buffer of size R must start on a K-bit boundary (that is, the K LSBs of the starting address of the circular buffer must be zeros), where K is an integer such that $2^{K} > R$. Since the value R must be loaded into the BK register, $K \ge N+1$. For example, a 31-word circular buffer must start at an address whose five LSBs are 0 (that is, xxx...x00000), and the value must be loaded into the BK register.

Note:

If the BK register has a value of 0, circular addressing is not performed. The effect will be the generation of a conventional linear address.



Figure 6–9. Register Relationships in Circular Addressing

- EB = effective base Н = high-order bits
- Ľ = new low-order bits

LSB = least significant bit

= location of the MSB equal to 1 in the BK register Ν

In circular addressing, index refers to the N LSBs of the auxiliary register selected, and step is the quantity being added to or subtracted from the auxiliary register. When you use circular addressing, follow two basic rules:

- The step used must be less than or equal to the block size and is treated as an unsigned integer.
- The first time the circular queue is addressed, the auxiliary register must be pointing to an element in the circular queue.

The algorithm for circular addressing is as follows:

 $\label{eq:steps} \begin{array}{l} \text{If } 0 \leq \text{index} + \text{step} < \text{BK};\\ \text{index} = \text{index} + \text{step}.\\ \text{Else if index} + \text{step} \geq \text{BK};\\ \text{index} = \text{index} + \text{step} - \text{BK}.\\ \text{Else if index} + \text{step} < 0;\\ \text{index} = \text{index} + \text{step} + \text{BK}. \end{array}$

Figure 6–10 shows how the circular buffer is implemented. It illustrates the relationship of the generated quantities and the elements in the circular buffer.

Figure 6–10. Circular Buffer Implementation



Figure 6–11 gives an example of the operation of circular addressing. Assuming that all registers are four bits, let $BK = 0110_2$ (block size of 6) and AR0 = 0000_2 (at least the 3 LSBs of AR0 should be 0). This example shows a sequence of modifications and the resulting value of AR0. It also shows how the pointer steps through the circular queue with a variety of step sizes (both incrementally and decrementally).

Figure 6–11. Circular Addressing Example

*AR0++(5)%	;	AR0	=	0	(0th value)
*AR0++(2)%	;	AR0	=	5	(1st value)
*AR0(3)%	;	AR0	=	1	(2nd value)
*AR0++(6)%	;	AR0	=	4	(3rd value)
*AR0%	;	AR0	=	4	(4th value)
*AR0	;	AR0	=	3	(5th value)



Circular addressing is especially useful for the implementation of FIR filters. Figure 6–12 shows one possible data structure for FIR filters. Note that the initial value of AR0 points to h(N-1), and the initial value of AR1 points to x(0). Circular addressing is used in the 'C4x code for the FIR filter shown in Example 6–22.

Figure 6–12. Data Structure for FIR Filters

	Impulse Response	Input Samples	
ARO —	h(N-1)	x(N-1)	
	h(N-2)	x(N-2)	
	•		
		·	
	h(2)	x(2)	
	h(1)	x(1)	
	h(0)	x(0)	— AR1

Example 6–22. FIR Filter Code Using Circular Addressing

* I1	nitiali	zation		
*				
	LDI	N,BK	;	Load block size.
	LDI	H,ARO	;	Load pointer to impulse response.
	LDI	X,AR1	;	Load pointer to bottom of input
*			;	sample buffer.
*				
TOP	LDF	IN, R3	;	Read input sample.
	STF	R3,*AR1++%	;	Store with other samples.
			;	and point to top of buffer.
	LDF	0,R0	;	Initialize R0.
	LDF	0,R2	;	Initialize R2.
*				
*	Filter	2		
*				
	RPTS	N - 1	;	Repeat next instruction.
	MPYF3	*AR0++%,*AR1	++?	s, R0
	ADDF3	R0,R2,R2	;	Multiply and accumulate.
	ADDF	R0,R2	;	Last product accumulated.
*				
	STF	R2,Y	;	Save result.
	В	TOP	;	Repeat.

Addressing Modes 6-31

6.9 Bit-Reversed Addressing

The 'C4x can implement fast Fourier transforms (FFT) with bit-reversed addressing. If the data to be transformed is in the correct order, the final result of the FFT is in bit-reversed order. To recover the frequency-domain data in the correct order, certain memory locations must be swapped. The bit-reversed addressing mode makes swapping unnecessary. The next time data must be accessed, it is accessed in a bit-reversed manner rather than sequentially. In the 'C4x, this bit-reversed addressing can be implemented with both the CPU and DMA.

For correct CPU (or DMA) bit-reverse operation, the base address of bit-reversed addressing must be located on a boundary of the size of the FFT table. The CPU bit-reverse operation can be illustrated by assuming an FFT table of size N = 2^{n} . When real and imaginary data are stored in separate arrays, the *n* LSBs of the base address must be zero, and IR0 must be equal to 2^{n-1} (half of the FFT size). When real and imaginary data are stored in consecutive memory locations (Re-Im-Re-Im), the n + 1 LSBs of the base address must be zero, and IR0 must be equal to 2ⁿ (FFT size).

For CPU bit-reversing, one auxiliary register (AR2 in this case) points to the physical location of a data value. When you add IR0 to this auxiliary register by using bit-reversed addressing, addresses are generated in a bit-reversed fashion (reverse carry propagation). The largest index for bit-reversed addressing is 0008 0000h; this index is treated as an unsigned integer.

To illustrate bit reversed addressing, assume 8-bit auxiliary registers. Let AR2 contain the value 0110 0000₂ (96₁₀). This is the base address of the data in memory. Let IR0 contain the value 0000 1000₂ (810). Example 6-23 shows a sequence of modifications of AR2 and the resulting values of AR2.

*AR2++(IR0)B;	AR2=	0110	0000	(Oth	val
*AR2++(IR0)B;	AR2=	0110	1000	(1st	val

Example 6–23. Bit-Reversed Addressing Example

*AR2++(IR0)B	;	AR2=	0110	0000	(Oth	value)
*AR2++(IR0)B	;	AR2=	0110	1000	(1st	value)
*AR2++(IR0)B	;	AR2=	0110	0100	(2nd	value)
*AR2++(IR0)B	;	AR2=	0110	1100	(3rd	value)
*AR2++(IR0)B	;	AR2=	0110	0010	(4th	value)
*AR2++(IR0)B	;	AR2=	0110	1010	(5th	value)
*AR2++(IR0)B	;	AR2=	0110	0110	(6th	value)
*AR2	;	AR2=	0110	1110	(7th	value)

Table 6–4 shows the relationship of the index steps and the four LSBs of AR2. You can find the four LSBs by reversing the bit pattern of the steps.

Table 6-4. Index Steps and Bit-Reversed Addressing

Step	Bit Pattern	Bit-Reversed Pattern	Bit-Reversed Step
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
8	1000	0001	1
9	1001	1001	9
10	1010	0101	5
11	1011	1101	13
12	1100	0011	3
13	1101	1011	11
14	1110	0111	7
15	1111	1111	15

Note:

Bit-reverse operation of the DMA coprocessor is covered in Chapter 11 of this user's guide and in the *TMS320C4x General-Purpose Applications User's Guide*.

Chapter 7

Page

Program Flow Control

The 'C4x provides a complete set of constructs that allow software and hardware control of the program flow. Software control includes repeats, branches, calls, traps, and returns. Hardware control includes interrupts. You can select the constructs best suited for your particular application.

Topic

7.1 Repeat Mode 7-2 7.2 Delayed Branches 7-9 7.3 Calls, Traps, Branches, Jumps, and Returns 7-12 7.4 Interrupts 7-15 7.5 Traps 7-24 7.6 DMA Interrupts 7-26 7.7 Reset 7-29

7.1 Repeat Mode

The repeat mode of the 'C4x can implement zero-overhead looping. For many algorithms, most execution time is spent in an inner kernel of code. Using the repeat modes allows these time-critical sections of code to be executed in the shortest possible time.

The 'C4x provides three instructions to support zero-overhead looping: RPTB (repeat a block of code), RPTBD (repeat a block of code delayed) and RPTS (repeat a single instruction):

- RPTB and RBTBD cause a block of code to be repeated a specified number of times.
- RPTS causes a single instruction to be repeated a number of times and reduces bus traffic by fetching the instruction only once.

RPTB and RPTS are four-cycle instructions; these four cycles of overhead are incurred only on the first pass through the loop. All subsequent passes through the loop are accomplished with zero cycles of loop overhead. RPTBD is a one-cycle instruction.

Three registers (RS, RE, and RC) control the updating of the program counter when it is updated in a repeat mode, as described in Table 7–1 below.

Table 7–1. Repeat-Mode Registers

Register	Function
RS	Repeat start address register. Holds the address of the first instruction of the code block to be repeated.
RE	Repeat end address register. Holds the address of the last instruc- tion of the code block to be repeated. RE should be greater than or equal to RS (see subsection 7.1.2).
RC	Repeat-count register. Contains one less than the number of times remaining for the code block to be repeated.

Correct operation of the repeat modes requires that all of the above registers and status register fields be initialized correctly. RPTB, RPTBD, and RPTS perform this initialization in slightly different ways (see subsection 7.1.3 and subsection 7.1.4 for more information).

7.1.1 Control Bits

Two bits are important to the operation of RPTB, RPTBD and RPTS:

- ☐ The **RM** (repeat-mode flag) bit in the status register specifies whether or not the processor fetches instructions during the repeat mode.
 - If RM = 0, fetches are not made in repeat mode.
 - If RM = 1, fetches are made in repeat mode.
- The S bit is internal to the processor and cannot be programmed, but this bit is necessary to fully describe the operation of RPTB, RPTBD, and RPTS.
 - If RM = 1 and S = 0, RPTB or RPTBD is executing. Program fetches occur from memory.
 - If RM = 1 and S = 1, RPTS is executing. After the first fetch (from memory), program fetches occur from the instruction register (IR).

7.1.2 Repeat-Mode Operation

Information in the repeat-mode registers and associated control bits is used to control the modification of the PC when instruction fetches are being made in repeat mode. The repeat modes compare the contents of the RE register (repeat end address register) with the program counter (PC) after the execution of each instruction. If they match and the repeat counter is nonnegative, the repeat counter is decremented, the PC is loaded with the repeat start address, and processing continues. The fetches and appropriate status bits are modified as necessary. Note that the repeat counter (RC) is never modified when the repeat-mode flag (RM) is 0.

The repeat counter should be loaded with a value one less than the number of times to execute the block; for example, an RC value of 4 would execute the block five times. The detailed algorithm for the update of the PC is shown in Example 7–1.

Notes:

- The maximum number of repeats occurs when RC = 8000 0000h. This results in 8000 0001h repetitions. The minimum number of repeats occurs when RC = 0. This results in one repetition.
- RE should be greater than or equal to RS (RE ≥ RS). Otherwise, the code will not repeat even though the RM bit remains set to 1.
- By writing a 0 into the repeat counter or writing 0 into the RM bit of the status register, you can stop the the loop before it completes.

Example 7–1. Repeat-Mode Control Algorithm

```
; If in repeat mode (RPTB or RPTS)
if S == 1
                                            ; If RPTS
  if first time through
                                            ; If this is the first fetch
     fetch instruction from memory
                                           ;Fetch instruction from memory
  else
                                            ; If not the first fetch
     fetch instruction from IR
                                           ;Fetch instruction from IR
  RC - 1 \rightarrow RC
                                            ;Decrement RC
  if RC < 0
                                           ; If RC is negative
                                           ;Repeat single mode completed
     0 \rightarrow \text{ST}(\text{RM})
                                            ;Turn off repeat mode bit
                                            ;Clear S
     0 \rightarrow S
     PC + 1 \rightarrow PC
                                            ;Increment PC
else if S == 0
                                           ; If RPTB
  fetch instruction from memory
                                           ;Fetch instruction from memory
  if PC == RE
                                           ; If this is the end of the block
     RC - 1 \rightarrow RC
                                           ;Decrement RC
  if RC \geq 0
                                           ; If RC is not negative
                                           ;Set PC to start of block
     RS \rightarrow PC
  else if RC < 0
                                            ; If RC is negative
     0 \rightarrow ST(RM)
                                            ; Turn off repeat mode bits
     0 \rightarrow S
                                            ;Clear S
     PC + 1 \rightarrow PC
                                            ;Increment PC
```

7.1.3 RPTB and RPTBD Instructions

The RPTB and RPTBD instructions repeat a block of code a specified number of times. RPTBD is a delayed form of the RPTB instruction that allows placing three instructions after it. These three instructions are not part of the block that is repeated, but they execute before the block repeat is started. This way, the pipeline remains full, and the RPTBD instruction can execute in one cycle.

The number of times to repeat the block is the RC (repeat count) register value plus one. Because the execution of RPTB and RPTBD does not load the RC, you must load this register yourself. The RC register must be loaded before the RPTB/RPTBD instruction is executed. *The RC register should not be loaded in the 3 instructions after RPTBD*. Example 7–2 shows a typical setup of the block repeat operation.

```
Example 7–2. RPTB Operation
```

LD 15,RC ; Load repeat counter with 15 RPTB ENDLOP; Execute the block of code ;from STLOOP to ENDLOP 16 times . ENDLOP All block repeats initiated by RPTB or RPTBD can be interrupted. However, interrupts are disabled during the execution of the three instructions following an RPTBD. None of the three instructions after the RPTBD instruction should modify the PC register or program flow. This restriction also applies to delayed branches, as explained in Section 7.2.

When RPTB *src* or RPTBD *src* execute, they perform a sequence of four operations:

- 1) Load the start address of the block into RS (repeat start address register).
 - For **RPTB**, this is the next address following the instruction: PC of RPTB + 1 \rightarrow RS
 - For **RPTBD**, this is the fourth address following the instruction: PC of RPTBD + $4 \rightarrow RS$
- 2) Load the end address of the block into RE (repeat end address register).
 - For RPTB, in *PC-relative mode*, the 24-bit *src* operand plus RS is the end address: *src* + PC of RPTB + 1 → RE
 - For RPTBD, in *PC-relative mode*, the 24-bit source operand plus RS is the end address: src + PC of RPTBD + 3 → RE
- 3) In *register mode*, the contents of the *src* register is the end address: contents of *src* register \rightarrow RE
- 4) Set the status register to indicate the repeat mode of operation. $1 \rightarrow RM$ status register bit (repeat mode flag)
- 5) Indicate that this is the repeat block mode of operation.
 0 → S bit (bit is internal to the processor and not programmable)

7.1.4 RPTS Instruction

A RPTS *src* instruction repeats the instruction following the RPTS (*src* +1) times. Repeats of a single instruction initiated by RPTS are not interruptible since RPTS fetches the instruction word only once and then keeps it in the instruction register for reuse. An interrupt in this situation would cause the instruction word to be lost. Refetching the instruction word from the instruction register reduces memory accesses and, in effect, acts as a one-word program cache. If you need a single instruction that is repeatable and interruptible, you can use the RPTB/RPTBD instruction.

When RPTS *src* is executed, a sequence of five operations occurs:

- 1) PC + 1 \rightarrow RS
- 2) $PC + 1 \rightarrow RE$
- 3) $1 \rightarrow RM$ (status register bit)
- 4) $1 \rightarrow S$
- 5) $src \rightarrow RC$ (repeat count register)

The RPTS instruction loads all registers and mode bits necessary for the operation of the single instruction repeat mode. Step 1 loads the start address of the block into RS. Step 2 loads the end address into the RE (end address of the block). Since this is a repeat of a single instruction, the start address and the end address are the same. Step 3 sets the status register to indicate the repeat mode of operation. Step 4 indicates that this is the repeat single-instruction mode of operation. Step 5 loads *src* into RC.

7.1.5 Repeat Mode Restriction Rules

Because the block repeat modes modify the program counter, other instructions cannot modify the program counter at the same time. Two rules apply:

Rule 1: The last instruction in the block (or the only instruction in a block of size one) cannot be a B*cond*, DB*cond*, CALL, CALL*cond*, TRAP*cond*, RETI*cond*, RETS*cond*, IDLE, RPTB, or RPTS. Example 7–3 shows an incorrectly placed standard branch.

Rule 2: None of the last four instructions from the bottom of the block (nor the only instruction in a block of size one) can be a *BcondD*, BRD, or *DBcondD*, RPTBD, LAJ, LAJ*cond*, LAT*cond*, B*cond*AF, B*cond*AT, or RETI*cond*D. Example 7–4 shows an incorrectly placed delayed branch.

If either of these rules is violated, the PC will be undefined.

Example 7–3. Incorrectly Placed Standard Branch

	LDI RPTB	15,RC ENDLOP	; ;	Load repeat counter with 15 Execute block of code
STLOOP			;	from STLOOP to ENDLOP 16 times
•				
•				
ENDLOP	BR	OOPS	;	This branch violates rule 1

```
Example 7–4. Incorrectly Placed Delayed Branch
```

```
LDI 15,RC ; Load repeat counter with 15

RPTB ENDLOP ; Execute block of code

; from STLOOP to ENDLOP 16 times

.

.

BRD OOPS ; This branch violates rule 2

ADDF

MPYF

ENDLOP SUBF
```

7.1.6 RC Register Value After Repeat Mode Completes

For the RPTB/RPTBD instruction, the RC register normally decrements to 0000 0000h, unless the block size is 1; in that case, it decrements to FFFF FFFFh. However, if the RPTB/RPTBD instruction with a block size of 1 has a pipeline conflict in the instruction being executed, the RC register decrements to 0000 0000h. Example 7–5 illustrates a pipeline conflict. Refer to Chapter 8 for pipeline information.

RPTS normally decrements the RC register to FFFF FFFFh. However, if the RPTS has a pipeline conflict on the last cycle, the RC register decrements to 0000 0000h.

In any case, the number of repetitions is always RC + 1, regardless of the final value of RC.

Example 7–5. Pipeline Conflict in a RPTB Instruction

EDC	.word	40000000h	;	Program is located in 4000000Fh
	LDP	EDC		
	LDI	@EDC,AR0		
	LDI	15 , RC	;	Load repeat counter with 15
	RPTB	ENDLOP	;	Execute block of code
ENDLOF	'LDI	*AR0,R0	;	The *ARO read conflicts with
			;	the instruction fetching.
			;	Then RC decrements to 0. If
			;	cache is enabled, RC decrements
			;	to FFFF FFFFh

7.1.7 Nesting Block Repeats

Block repeats (RPTB and RPTBD) are nestable. Because all of the control of a block repeat is defined by the RS, RE, RC, and ST registers, these registers must be saved and stored to nest block repeats. For example, if you write an interrupt service routine that requires the use of RPTB or RPTBD, it is possible that the interrupt associated with the routine may occur during another block repeat. The interrupt service routine can check the RM bit to determine whether the block repeat mode is active. If RM is set, the interrupt routine should save ST, RS, RE and RC, in this order. The interrupt routine can then perform a block repeat. Before returning from the interrupted routine, the interrupt routine should restore RC, RE, RS, and ST, in this order. If the RM bit is not set, you do not need to save and restore these registers.

The RPTS instruction can also be used in a block repeat loop if the proper registers are saved.

Because the program counter is modified at the end of the loop according to the contents of registers RS, RE, and RC, no operation should attempt to modify the repeat counter or the program counter to a different value at the end of the loop.

It takes four cycles of overhead to save and restore these registers. Hence, sometimes, it may be more economical to implement a nested loop by the more traditional method of using a register as a counter and then using a delayed branch, rather than by using the nested repeat block approach. Often, implementing the outer loop as a counter and the inner loop as a RPTB/RPTBD instruction produces the fastest execution.

Note:

The order in which the registers are saved/restored is important to guarantee correct operation. The ST register should be restored last, after the RC, RE, and RS registers. ST should be restored after restoring RC, because the RM bit cannot be set to one if the RC register is 0 or -1. For this reason, if you execute a POP ST instruction (with ST(RM) = 1) while RC = 0, the POP instruction recovers all of the ST register bits except the RM bit, which stays at 0 (repeat mode disabled). Also, RS and RE should be correctly set before you activate the repeat mode.

Section 1.7, *Repeat Modes*, in the *TMS320C4x General-Purpose Applications User's Guide* contains examples of how to use repeat-mode instructions.

7.2 Delayed Branches

The 'C4x offers two main types of branches: standard and delayed.

Standard branches empty the pipeline before performing the branch; this guarantees correct management of the program counter and results in a 'C4x branch taking four cycles. Included in this class are standard branches (B*cond*), repeats, calls, returns, and traps.

Delayed branches do not empty the pipeline but guarantee that the next three instructions will execute before the program counter is modified by the branch. The result is a branch that requires only a single cycle, thus making the speed of the delayed branch very close to speed of the optimal block repeat modes of the 'C4x. However, unlike block repeat modes, delayed branches can be used in situations other than looping. Every delayed branch has a standard branch counterpart that is used when a delayed branch cannot be used.

Conditional delayed branches use the conditions, reflected in the status register, that existed at the end of the instruction preceding the branch. They do not depend upon the instructions following the delayed branch. The execution time of a conditional delayed branch instruction is the same regardless of whether or not the branch is taken.

When a delayed branch is fetched, it remains pending until the three instructions that follow are executed. None of the three instructions immediately after a delayed branch can be any of the following:

Bcond	DB <i>cond</i>	LAJ	RETS <i>cond</i>
B <i>cond</i> D	DB <i>cond</i> D	LAJ <i>cond</i>	RPTB
B <i>cond</i> AF	CALL	LAT <i>cond</i>	RPTBD
B <i>cond</i> AT	CALLcond	RETI <i>cond</i>	RPTS
BR	IDLE	RETI <i>cond</i> D	TRAP <i>cond</i>
BRD			

This restriction also applies to the RPTBD instruction, covered in subsection 7.1.3.

Delayed branches disable interrupts until the three instructions following the delayed branch are completed. This is independent of whether or not the branch is taken.

Incorrectly used delayed branches can leave the PC undefined. Example 7–6 illustrates an incorrectly-placed delayed branch.

Example 7–6. Incorrectly Placed Delayed Branches

```
B1:BD L1
NOP
NOP
B2:B L2 ; This branch is incorrectly placed
NOP
NOP
NOP
.
```

Sometimes, a branch is necessary for the program flow when fewer than three instructions can be placed after a delayed branch. For faster execution, it is still advantageous to use a delayed branch. This is shown in Example 7–7, with a NOP taking the place of the third unused instruction. The tradeoff is more instruction words for less execution time.

Example 7–7. Delayed Branch Execution

```
TITLE DELAYED BRANCH EXECUTION
      LDF*+AR1(5),R2 ; Load contents of memory to R2
      BGED SKIP ; If loaded number >=0, branch
                        (delayed)
      LDFN R2,R1
           R2,R1 ; If loaded number <0, load it to R1
3.0,R1 ; Subtract 3 from R1
      SUBF
      NOP
                     ; Dummy operation to complete
                        delayed branch
      MPYF 1.5,R1
                    ; Continue here if loaded number < 0
      .
      .
SKIP
      LDF
            R1,R3
                      ;Continue here if loaded number >=0
```

There are two types of delayed branches: branches without annulling and branches with annulling.

7.2.1 Delayed Branches Without Annulling

Delayed branches without annulling do not empty the pipeline but guarantee that the next three instructions execute before the program counter is modified by the branch. The delayed branches without annulling are B*cond*D, BRD, and DB*cond*D.

7.2.2 Delayed Branches With Annulling

Delayed branches with annulling may conditionally annul the next three instructions. The delayed branches with annulling are B*cond*AT and B*cond*AF:

BcondAF

If the condition is true, the B*cond*AF instruction executes the three instructions following the branch and then branches. If the condition is false, the processor does not take the branch and it annuls the effects of the execute phase of the first following instruction and the effects of the read and execute phases of the second and third following instructions.

BcondAT

If the condition is true, the B*cond*AT instruction causes a branch and annuls the effects of the execute phase of the first following instruction and the effects of the read and execute phases of the second and third following instructions. If the condition is false, the instruction causes the execution of the three instructions following the branch and does not cause a branch.

7.3 Calls, Traps, Branches, Jumps, and Returns

Calls and traps can execute a subroutine or function while providing a return to the calling routine.

The CALL, CALL*cond*, and TRAP*cond* instructions store the value of the PC on the stack before changing the PC's contents. The RETS*cond* or RETI*cond* (standard or delayed) instructions use the value on the stack to return execution from traps and calls.

CALL is a four-cycle instruction, while CALL*cond* and TRAP*cond* are fivecycle instructions. 'C4x delayed instructions LAJ, LAJ*cond*, and LAT*cond* provide equivalent functionality, respectively, but in a single cycle.

- CALL places the next PC value on the stack and places the *src* (source) operand into the PC. The *src* is a 24-bit PC-relative or register value. Figure 7–1 shows CALL response timing.
- CALLcond is similar to the CALL instruction (above) except for two differences:
 - It executes only if a specific condition is true (the 20 conditions including unconditional — are listed in Section 14.2 on page 14-12).
 - The src is either a 24-bit PC-relative displacement or in register addressing mode.
- TRAPcond executes only if a specific condition is true (same conditions as for the CALLcond instruction). When it executes, a four-step sequence occurs:
 - The values of the GIE and CF status register bits are saved into the PGIE and PCF status register bits.
 - 2) Interrupts are disabled (GIE = 0) and the cache is frozen (CF bit = 0).
 - 3) The next PC value is stored on the stack.
 - The specified vector is retrieved from the trap vector table and is loaded into the PC. The vector address corresponds to a trap number in the instruction.

Using RETIcond or RETIcondD to return re-enables interrupts if the status register's GIE bit was set previously and recovers the previous CF bit.

RETScond returns execution from any of the above three instructions by popping the top of the stack to the PC. For RETScond to execute, the specified condition must be true. The conditions are the same for RETScond as for the CALLcond instruction.

- RETIcond returns from traps or calls in the same way that RETScond does with the addition that RETIcond also copies the PGIE and PCF bit values into the GIE and CF bits of the status register. The conditions for RETIcond are the same as for the CALLcond instruction.
- RETIcondD returns from traps or calls in the same way that RETIcond does with the addition that RETIcondD first executes the three instructions immediately following RETIcondD. The conditions for RETIcondD are the same as for the CALLcond instruction.
- Link and jump (LAJ), link and jump conditional (LAJcond), and link and trap conditional (LATcond) each provide a return address in extendedprecision register R11.
 - After it executes the three instructions that follow it, LAJ jumps to an address derived by a 24-bit PC-relative addressing mode (see subsection 6.6 for more information).
 - The LAJcond destination address is either PC-relative (a displacement) or the contents of a specified register. If the condition is true, LAJcond first executes the three instructions following the LAJcond before making the jump. If the condition is not true, execution continues immediately after the LAJcond instruction.
 - After it executes the three instructions that follow it, LAT*cond* calls one of the 512 available trap vectors pointed to by the trap vector table pointer (see Section 3.2, on page 3-17, for more information about the TVTP).

Functionally, calls and traps accomplish the same task—a subfunction is called and executed, and control is then returned to the calling function. Traps offer two advantages over calls:

- Interrupts are automatically disabled when a trap is executed. This allows critical code to execute without risk of being interrupted. Thus, traps are usually terminated with a RETI*cond* or RETI*cond* D instruction to re-enable interrupts if the status register GIE bit was set previously.
- You can use traps to indirectly call functions. This is particularly beneficial when a kernel of code contains the basic subfunctions to be used by applications. In this case, you can modify the functions in the kernel and relocate them without recompiling each application.


Figure 7–1. CALL Response Timing

7.4 Interrupts

The 'C4x supports multiple internal and external interrupts, which can be used for a variety of applications. Internal interrupts are generated by the DMA controller, the timers, and the communication ports. The five external interrupt pins include four external maskable interrupt pins ($\overline{IIOF0}$ – $\overline{IIOF3}$) and one non-maskable interrupt (\overline{NMI}) pin. Interrupts can be sent to both the CPU and the DMA controller.

Interrupts on the 'C4x are automatically prioritized. This allows interrupts that occur simultaneously to be serviced in a predefined order.

This section discusses the operation of these interrupts. Additional information regarding internal interrupts can be found in Section 12.6, *Coordinating Communication Ports with the CPU and DMA Processor*, on page 12-17, Section 11.10, *DMA and Interrupts*, on page 11-42, and Chapter 13, *Timers*. See Section 7.6, *DMA Interrupts*, on page 7-26, for more information about interrupts to the DMA controller.

7.4.1 Interrupt Vector Table and Prioritization

The interrupt vector table (IVT) shown in Figure 7–2 contains the interrupt vectors. An interrupt vector is an address of an interrupt service routine that should start executing when an interrupt is received. The IVT table must be placed on a 512-word memory boundary. The table location is determined by the value that is stored in the IVTP register (see Section 3.2, *CPU Expansion Register File*, on page 3-17).

Prioritization means that an interrupt in a higher position in the interrupt vector table (Figure 7–2) is serviced before one in a lower position *when both are received in the same clock cycle or when two previously received interrupts are waiting to be serviced*. It *does not* mean, for example, that $\overline{IIOF3}$ must wait until service routines for $\overline{IIOF2}$, $\overline{IIOF1}$, and $\overline{IIOF0}$ are completed (when ST(GIE) = 1).

The priority of interrupts is handled by the CPU according to the interrupt vector table. Priority is set according to position in the table — those with displacements closest to the IVTP base address are higher in priority (i.e., NMI is higher than TINT0, which is higher than $\overline{IIOF0}$, etc.). Note that interrupt TINT0 is located at IVTP + 2, while the TINT1 vector is located at IVTP + 2Bh after the communication port and DMA coprocessor interrupts.



Figure 7–2. Interrupt-Vector Table (IVT)

Notes: 1) Reserved for the reset vector. See Table 7–4.

- 2) NMI (the nonmaskable interrupt) is discussed in subsection 7.4.5.
- Timer interrupts TINT0 and TINT1 are enabled by the IIE register (subsection 3.1.9, page 3-11) and monitored at the IIF register (subsection 3.1.10, page 3-13).
- 4) External pins IIOF0—IIOF3 are programmed in the IIF register (subsection 3.1.10, page 3-13).
- The communication port I/O buffers full/empty/ready interrupts are enabled by the IIE register and are also described in Figure 12–4, on page 12-8, (OUTPUT LEVEL and INPUT LEVEL bits).
- Interrupts from the DMA are enabled at the IIE register and DMA channel control register at bits TCC and AUX TCC (see Figure 11–2, on page 11-8, for bit descriptions).
- In the 'C44, the interrupts for communication ports 0 and 3 are active. If you enable them with the IE bit, the ISR will be executed.

7.4.2 CPU Interrupt Control Bits

Three CPU registers contain bits used to control CPU interrupt operation:

- The CPU status register (ST). The CPU global interrupt enable bit (GIE), located in the ST, controls all maskable CPU interrupts. When this bit is set to 1, CPU interrupts are globally enabled. When this bit is cleared to 0, all CPU interrupts are disabled (except NMI, the nonmaskable interrupt). Refer to subsection 3.1.7, Status Register (ST), on page 3-5.
- Internal interrupt enable register (IIE). The IIE is used to enable CPU internally-generated interrupts (from timers, communication ports, and DMA channels). See subsection 3.1.9, CPU Internal Interrupt Enable Register (IIE), on page 3-11, for more information.
- IIOF flag register (IIF). The IIF contains interrupt flag bits and bits to determine the function of the external-interrupt pins (IIOF0 IIOF3).

The IIF Register

When an external interrupt or most of the internal interrupts are received, a corresponding bit in the IIF register is set to 1. The only internally generated interrupts that do not have a flag bit in the IIF register are the communication port interrupts.

When the CPU services an interrupt that has an interrupt flag bit in the IIF register, or when the DMA controller latches this type of interrupt into a DMA internal signal, this flag bit is cleared by the internal interrupt acknowledge signal. However, for level-triggered interrupts, if IIOFn is still low when the interrupt acknowledge signal occurs, the interrupt flag bit is cleared for only one cycle and then set to 1 again. For this reason, it is theoretically possible that, depending on when the IIF register is read, the interrupt flag bit may be zero, even though IIOFn is low. After reset, zero is written to the interrupt flag register, thereby clearing all pending interrupts.

The IIF register bits can be read or written under software control. This provides access to the \overline{IIOFx} pins, which can be treated as general-purpose I/O or as interrupt pins. For example, if at the IIF register, FUNCx = 0 (I/O pin) and TYPEx = 1 (output pin), then by writing into the FLAGx bit, you can also write to the external pin \overline{IIOFx} . If FUNCx = 1 (interrupt pin), writing a 1 to the IIF register FLAGx bit has the same effect as an incoming interrupt received on the corresponding pin. In this way, all interrupts can be triggered and/or cleared through software. Since the interrupt bits also can be read, the interrupt pins can be polled in software when an interrupt-driven interface is not required.

Internal interrupts operate in a similar manner. In the IIF register, the bit corresponding to an internal interrupt (e.g., TINT0, TINT1) can be read and written to through software. Writing a 1 sets the interrupt latch, and writing a 0 clears it. All internal interrupts are one H1/H3 cycle in length. If any previous bit values of the IIF register need to be preserved, a modification to IIF should be performed with logic operations (AND, OR, etc), directly to the IIF register.

Figure 7–3. IIF Register Modification

correct	incorrect
LDI @MASK,R0	LDI IIF, R1
AND RO, IIF	AND @MASK, R1
	LDI R1, IIF

7.4.3 Interrupt Processing

For an interrupt to occur, at least two conditions must be met:

- All interrupts must be enabled globally by setting the GIE bit to 0 in the CPU status register (ST).
- The interrupt must be enabled by setting the corresponding bit in the IIE register.

The CPU interrupt processing cycle (shown in Figure 7–4) involves several events. The corresponding interrupt flag in the IIF register is cleared, the values of the GIE and CF status register bits are preserved, the cache is frozen (CF = 1), interrupts are globally disabled (GIE = 0), and the CPU completes all fetched instructions. Then, the interrupt vector is fetched and loaded into the PC, and the CPU continues execution of the first instruction in the interrupt service routine (ISR). When you use RETI*cond* or RETI*cond*D to return from the interrupt service routine, the previous GIE and CF bit values are recovered.

If you wish to make the interrupt service routine interruptible, you can set the GIE bit to 1 after entering the ISR. In addition, you can enable the cache. Be aware that because the PGIE and PCF status register bits are one deep, they preserve only the previous GIE and CF bits.

Note:

The GIE, and CF are preserved and loaded with new values after the completion of the last instruction that was fetched before the interrupt was flushed. This guarantees later restoration of correct flag values.

Figure 7-4. CPU Interrupt Processing



CPU interrupts (including NMI) are only acknowledged (responded to by the CPU) on instruction fetch boundaries. If instruction fetches are halted because of pipeline conflicts or when an RPTS loop is executing, CPU interrupts are not acknowledged until the next instruction fetch.

The interrupt acknowledge (IACK) instruction can be used to signal externally that an interrupt has been serviced. If external memory is specified in the operand, IACK drives the IACK pin and performs a dummy read. The read is performed from the address specified by the IACK instruction operand. IACK is typically placed in the early portion of an interrupt service routine. However, depending on your application, it may be better suited at the end of the interrupt service routine or at another location. You are not required to use the IACK instruction in interrupt service routines.

Note the following situations:

- Interrupts are disabled during a RPTS and during a delayed branch (until the 3 instructions following a delayed branch are completed). Interrupts are held until after the branch.
- When an interrupt occurs, instructions currently in the decode and read phases continue regular execution. This is not the case for an instruction in the fetch phase:
 - If the interrupt occurs in the first cycle of the fetch of an instruction, the fetched instruction is discarded (not executed), and the address of that instruction is pushed to the top of the system stack.
 - If the interrupt occurs after the first cycle of the fetch (in the case of a multicycle fetch due to wait states), that instruction is executed, and the address of the next instruction to be fetched is pushed to the top of the system stack.
 - If no program fetch is occurring, then no new fetch is performed.

7.4.4 CPU Interrupt Latency

CPU interrupt latency, defined as the time from the acknowledgement of the interrupt to the execution of the first instruction of the interrupt service routine (ISR), is at least 8 cycles. This is explained in Table 7–2 where the interrupt is treated as an instruction, assuming that all the instructions are single-cycle instructions.

Table 7–2. Interrupt Latency

Cycle	Description	Fetch	Decode	Read	Execute
1	Recognize interrupt in single-cycle fetched (prog a+1) instruction.	prog a+1	prog a	prog a–1	prog a–2
2	Temporarily disable interrupt until GIE is cleared. Clear the corresponding IIF flag (if applicable).	_	interrupt	prog a	prog a–1
3	Read the interrupt vector table.	_	—	interrupt	prog a
4	Store return address to stack; save the GIE bit into PGIE and CF into PCF. Then, clear the GIE bit and set the CF bit to 1.	_	_	_	interrupt
5	Pipeline begins to fill with ISR instruction.	isr1	_	_	_
6		isr2	isr1	—	—
7		isr3	isr2	isr1	—
8	Execute first instruction of interrupt service routine.	isr4	isr3	isr2	isr1

7.4.5 External Interrupts

The five external interrupt pins include four external maskable interrupt pins ($\overline{IIOF0}-\overline{IIOF3}$) and one nonmaskable interrupt (\overline{NMI}) pin.

The four external maskable interrupts ($\overline{IIOF0}$ - $\overline{IIOF3}$) are enabled at the IIF register (subsection 3.1.10 page 3-13) and are synchronized internally. They are sampled on the falling edge of H1 and passed through a series of H1/H3 delays internally. Once synchronized, the interrupt input will set the corresponding interrupt flag register (IIF) bit if the interrupt is active. The list below shows the external interrupts and their corresponding interrupt vectors:

IIOF Pin and Interrupt	Interrupt Vector Location
IIOF0	IVTP + 003h
IIOF1	IVTP + 004h
IIOF2	IVTP + 005h
IIOF3	IVTP + 006h

These interrupts are prioritized by the selection of one over the other if both come on the same clock cycle (IIOF0 the highest, IIOF1 next, etc.). When an interrupt is taken, the status register ST(GIE) bit is reset to 0, disabling any other incoming interrupt (except NMI). This prevents any other interrupt (IIOF0–IIOF3) from assuming program control until the ST(GIE) bit is set back to 1. In addition, the ST(GIE) bit is saved into ST(PGIE) and the ST(CF) bit into ST(PCF). On a return from an interrupt routine, the RETI and RETI*cond* instructions place the value that is in the ST(PGIE) bit into the ST(GIE) bit and ST(PCF) bit into the ST(CF) bit, returning them to their previous values.

External interrupts can be either *edge- or level-triggered*, depending on how the TYPE fields are set in the IIF register (see subsection 3.1.10, *IIOF Flag Register (IIF)*, on page 3-13, for more information about the IIF).

For an edge-triggered interrupt to be detected by the 'C4x, the external pin must transition from 1 to 0. And then, it needs to be held low for at least one H1/H3 cycle (but it could be held low longer).

For a level-triggered interrupt to be detected by the 'C4x, the external pin needs to be held low for between one and two cycles ($1 \le$ low-pulse width \le 2). If the interrupt is held low for more than two cycles, more than one interrupt might be recognized. There is no need to provide an edge in this case.

Note:

Level-triggered interrupts are unlatched. The 'C4x will only detect them if the low-level is present during a fetch-to-decode pipeline transition. This means that during a pipeline halt, the level-triggered interrupts might be missed even if they are held low between one and two cycles. This is not the case for an edge-triggered interrupt because they are latched (they will get recognized regardless if the pipeline is halted).

NMI

The nonmaskable interrupt, NMI (an incoming low on pin AJ5, signal NMI), is not masked by the ST(GIE) bit. Even though the NMI is nonmaskable, its processing is temporarily postponed during delayed branches and multicycle CPU operations. NMI is a negative-going, edge-triggered, latched interrupt.

Take special care when using an NMI as a second level interrupt. When the 'C4x services an interrupt, interrupts are disabled except for the NMI. This creates a problem because the ST register may end up with the wrong value if the NMI is executed before the first level ISR that preserves the ST register's value.

The TMS320C44 and the TMS320C40 (revision 5.0 and greater) has a software-configurable feature that allows the forcing ready of the internal peripheral bus when the $\overline{\text{NMI}}$ signal is asserted. This NMI bus-grant feature is enabled when bits 18 and 19 in the status register (ST) are set to 10_2 . When enabled, a peripheral bus-grant signal is generated on the falling edge of $\overline{\text{NMI}}$. If $\overline{\text{NMI}}$ is asserted and this feature is not enabled, the CPU stalls on an access to the peripheral bus if the bus is not ready. A stall condition occurs when writing to a full output FIFO or reading an empty input FIFO. This feature is useful in correcting communication-port errors when used in conjunction with the communication-port software-reset feature.

7.5 Traps

A trap is the equivalent of a software-triggered interrupt. In the 'C4x, traps and interrupts are treated identically, except in the way in which they are initialized.

7.5.1 Initialization of Traps and Interrupts

Traps and interrupts are initialized differently in the 'C4x.

Traps are always triggered by a software mechanism, by the TRAP*cond* (conditional trap) and LAT*cond* (link and trap conditionally delayed) instructions.

Interrupts are always triggered by hardware events (for example, by external interrupts, DMA interrupts, or communication channel interrupts).

These GIE bit in the ST register and the mask bits in the IIE do not apply to traps.

7.5.2 Operation of Traps

Figure 7–5 shows the general flow of traps (and also of interrupts).



The **RETI***cond* and **RETI***cond***D** instructions manipulate the status flags as shown in block (3) in Figure 7–5. RETI*cond*/RETI*cond*D provides a return/ delayed return from a trap or interrupt.

Figure 7–5. Flow of Traps

In general, you should not directly modify the PGIE or PCF status register bits except when putting the status register on a stack for recursive interrupts or traps.

The 'C4x supports 512 different traps. When a TRAPcond n or LATcond n instruction is executed, the 'C4x jumps to the address stored in the memory location pointed to by TVTP + n, where TVTP is the Trap Vector Table Pointer register. The 32–bit TVTP register is essentially the base address for the trap-vector table (TVT) in memory. This table, shown in Figure 7–6, contains the addresses of the trap service routines that are executed by the CPU.

Figure 7–6. Trap Vector Table (TVT)

TVTP + 000h	TRAP0
TVTP + 001h	TRAP1 to
TVTP + 1FEh	TRAP510
TVTP + 1FFh	TRAP511

As with the interrupt vector table (IVT), the trap vector table (TVT) must begin on a 512-word memory boundary. The TVT pointer register (TVTP) points to the beginning of the TVT. See Section 3.2, *CPU Expansion Register File*, on page 3-17, for more information about the TVTP.

The TRAP or LAT*cond* instructions can be used to generate a trap and manipulate the status flags as shown in block (1) in Figure 7–5. LAT*cond* (link and trap conditionally) provides a single-cycle trap that is very useful for error detection and correction.

Note:

Because LAT*cond* is a delayed instruction, the three instructions following LAT*cond* should not modify the GIE or CF status register bits (this could result in storing incorrect values of these two bits).

7.5.3 Overlapping the Trap and Interrupt Vector Tables

The interrupt and trap vector tables can share the same 512-byte space in memory. In this configuration, you can place trap vectors where there are no interrupt vectors. For example, since interrupt vector 02Ch is unused, you could place a trap vector at IVTP+02Ch (which is also TVTP+02Ch if the tables overlap) and then call that trap by specifying 02Ch in the TRAP instruction.

7.6 DMA Interrupts

Interrupts can trigger DMA read and write operations. This is called DMA synchronization. The DMA interrupt processing cycle is similar to that of the CPU. After the pertinent interrupt flag is cleared, the DMA coprocessor proceeds according to the status of the SYNC bits in the DMA coprocessor global control register.

If the interrupt in the DMA Interrupt Enable (DIE) register is enabled, the interrupt controller automatically latches the interrupt and saves it for future DMA use. In the case of the flag interrupts (timer, external interrupt), the IIF flags are cleared when the interrupt controller latches the interrupt, not when the DMA responds to it. Even if the DMA has not been started, the interrupt latch occurs, except when the start bits in the DMA control register have the reset value 00₂ in START (AUX START) bits. DMA reset clears the interrupt internal latch.

7.6.1 DMA Interrupt Control Bits

Two registers contain bits used to control DMA interrupt operation:

- DMA interrupt enable register (DIE). All DMA interrupts are controlled by bits in the DIE and by the SYNC bits of the DMA channel control registers (described in Figure 11–2). The DMA interrupts are not dependent upon ST(GIE) and are local to the DMA.
- ☐ The DMA channel control register. Each DMA coprocessor channel uses a channel control register to determine its mode of operation. This register is shown in Figure 11–2.

The DIE is broken into six subfields that determine which interrupts can be used to control the synchronization for each of the six DMA channels. For example, the bits in these each of these fields allow you to select whether a DMA channel is synchronized to a communication port, a timer, or an external interrupt pin.

See subsection 3.1.8, *DMA Coprocessor Interrupt Enable Register (DIE)*, on page 3-8, for a description of the DIE.

7.6.2 DMA Interrupt Processing

Figure 7–7 shows the general flow of interrupt processing by the DMA coprocessor.

Figure 7–7. DMA Interrupt Processing



For more information about DMA interrupts, see Section 11.10, DMA and Interrupts.

7.6.3 CPU/DMA Interrupt Interaction

The 'C4x DMA coprocessor is not affected by the processing of CPU interrupts, even when the DMA is using interrupts for synchronization of transfers. In addition, the DMA is be affected, even when pipeline fetches are halted.

The 'C4x allows the CPU and DMA controller to respond to and process interrupts in parallel. Figure 7–8 shows the sequence of events in interrupt processing for both the CPU and the DMA controller; for the exact sequence of events, see Table 7–2.

It is therefore possible to interrupt the CPU and DMA coprocessors simultaneously with the same or different interrupts and, in effect, synchronize their activities. However, because the DMA coprocessor and CPU share the same set of interrupt flags, in some instances the DMA coprocessor can clear an interrupt flag before the CPU can respond to it. For example, if CPU interrupts are disabled or if instruction fetches have been halted, the DMA can latch the interrupt and thus clear the associated interrupt flag. If the interrupt is enabled in the DIE register, the CPU will never be able to "steal" a DMA interrupt, because the DMA responds to an interrupt as fast as or faster than the CPU.

Figure 7–8. Parallel CPU and DMA Interrupt Processing



7.7 Reset

The 'C4x supports a nonmaskable external reset signal (RESET), which is used to perform system reset. This section discusses the reset operation.

After powerup, the state of the 'C4x processor is undefined. You can use the RESET signal to put the processor in a known state. This signal must be asserted low for 10 or more H1 clock cycles to guarantee a system reset (See Chapter 1, *Processor Initialization*, in the *TMS320C4x General-Purpose Applications User's Guide* for the recommended reset circuit). H1 is an output clock signal generated by the 'C4x.

Reset affects several aspects of 'C4x operation:

- Some device pins
- Some device registers
- Program execution

7.7.1 Reset's Effects on Pin States

Reset affects the other pins on the device in either a synchronous or an asynchronous manner. The synchronous reset is gated by the 'C4x's internal clocks. The asynchronous reset directly affects the pins and is faster than the synchronous reset. Reset timing details are included in the 'C4x data sheets.

Table 7–3 shows the state of the 'C4x's pins during $\overline{\text{RESET}} = 0$ and after $\overline{\text{RE-SET}}$ goes back to 1. Each pin is described according to whether the pin is reset synchronously or asynchronously.

(a) Clock (4 pins)								
Signal	Pins	I∕O§	Type†	Description				
H1	1	0	S	Begins clocking when RESET makes a 1-to-0 transition				
H3	1	0	S	Begins clocking when RESET makes a 1-to-0 transition				
X1	1	0	_	No effect				
X2/CLKIN	1	Ι	_	No effect				

Table 7–3. Pin States At System Reset

 † A = Asynchronous, S = Synchronous

[‡] Recommended decoupling capacitors are one multiple 0.1 μF and 4.7 μF around the device. Number depends on specific board noise conditions.

Table 7–3. Pin States After System Reset (Continued)

(b) Communication Port 0 Interface (12 pins)

Signal	Pins	I/O§	Type†	Description
C0D(7-0)	8	I/O	S	Set to undefined value
CACK0	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high
CRDY0	1	I/O	А	Set to high-impedance
CREQ0	1	I/O	А	Set to high-impedance
CSTRB0	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high

(c) Communication Port 1 Interface (12 pins)

Signal	Pins	I∕O§	Type [†]	Description
C1D(7-0)	8	I/O	S	Set to undefined value
CACK1	1	I/O	А	Set high-impedance when reset goes low and then set to one when reset goes high
CRDY1	1	I/O	А	Set to high-impedance
CREQ1	1	I/O	А	Set to high-impedance
CSTRB1	1	I/O	А	Set high-impedance when reset goes low and then set to one when reset goes high

(d) Communication Port 2 Interface (12 pins)

Signal	Pins	I∕O§	Type [†]	Description
C2D(7-0)	8	I/O	S	Set to undefined value
CACK2	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high
CRDY2	1	I/O	А	Set to high-impedance
CREQ2	1	I/O	А	Set to high-impedance
CSTRB2	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high

 † A = Asynchronous, S = Synchronous

[‡] Recommended decoupling capacitors are one multiple 0.1 μF and 4.7 μF around the device. Number depends on specific board noise conditions.

Table 7–3. Pin States After System Reset (Continued)

(e) Communication Port 3 Interface (12 pins)

Signal	Pins	I/O§	Type [†]	Description
C3D(7-0)	8	I/O	S	Set to high-impedance
CACK3	1	I/O	А	Set to high-impedance
CRDY3	1	I/O	А	Set high-impedance when reset goes low and then set to one when reset goes high
CREQ3	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high
CSTRB3	1	I/O	А	Set to high-impedance

(f) Communication Port 4 Interface (12 pins)

Signal	Pins	I∕O§	Type [†]	Description
C4D(7-0)	8	I/O	S	Set to high-impedance
CACK4	1	I/O	А	Set to high-impedance
CRDY4	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high
CREQ4	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high
CSTRB4	1	I/O	А	Set to high-impedance

(g) Communication Port 5 Interface (12 pins)

Signal	Pins	I∕O§	Type [†]	Description
C5D(7-0)	8	I/O	S	Set to high-impedance
CACK5	1	I/O	А	Set to high-impedance
CRDY5	1	I/O	A	Set high-impedance when reset goes low and then set to one when reset goes high
CREQ5	1	I/O	А	Set high-impedance when reset goes low and then set to one when reset goes high
CSTRB5	1	I/O	А	Set to high-impedance

 † A = Asynchronous, S = Synchronous

[‡] Recommended decoupling capacitors are one multiple 0.1 μ F and 4.7 μ F around the device. Number depends on specific board noise conditions.

(h) Emulation (7 pins)									
Signal	Pins	I∕O§	Type [†]	Description					
EMU0	1	I/O	-	Undefined					
EMU1	1	I/O	-	Undefined					
ТСК	1	I	_	No effect					
TDI	1	I	_	No effect					
TDO	1	0	_	No effect					
TMS	1	I	_	No effect					
TRST	1	Ι	_	No effect					

Table 7–3. Pin States After System Reset (Continued)

(i) Global Bus External Interface (80 pins)

Signal	Pins	I∕O§	Type [†]	Description
A(30-0)	31	O/Z	S	Set to high-impedance
AE	1	I	_	No effect
CE0	1	I	_	No effect
CE1	1	I	_	No effect
D(31-0)	32	I/O/Z	S	Set to high-impedance
DE	1	I	_	No effect
LOCK	1	0	S	Set to one
PAGE0	1	O/Z	S	Set to zero
PAGE1	1	O/Z	S	Set to zero
RDY0	1	I	_	No effect
RDY1	1	I	_	No effect
R/W0	1	O/Z	S	Set to one
R/W1	1	O/Z	S	Set to one
STAT(3-0)	4	0	S	Set to all ones
STRB0	1	O/Z	S	Set to one
STRB1	1	O/Z	S	Set to one

[†]A = Asynchronous, S = Synchronous

 \ddagger Recommended decoupling capacitors are multiple 0.1 μF and 4.7 μF around the device. Number depends on specific board noise conditions. § I=Input, O=Output, Z=High-impedance state.

Signal	Pins	I∕O§	Type [†]	Description
LA(30-0)	31	O/Z	S	Placed in high-impedance state
LAE	1	I	_	Reset has no effect
LCE0	1	I	_	Reset has no effect
LCE1	1	I	_	Reset has no effect
LDE	1	I	_	Reset has no effect
LLOCK	1	0	S	Set to one
LPAGE0	1	O/Z	S	Set to zero
LPAGE1	1	O/Z	S	Set to zero
LRDY0	1	I	_	Reset has no effect
LRDY1	1	I	_	Reset has no effect
LR/W1	1	O/Z	S	Set to one
LSTAT(3-0)	4	0	S	Set to all ones
LSTRB0	1	O/Z	S	Set to one
LSTRB1	1	O/Z	S	Set to one

Table 7–3. Pin States After System Reset (Continued)

(j) Local Bus External Interface (80 pins)

(k) Interrupts, I/O Flags, Reset, Timer (12 pins)

Signal	Pins	I∕O§	Type [†]	Description
IACK	1	Ι	S	Set to one
IIOF(0-3)	4	I/O	А	Set to high-impedance
NMI	1	I	_	No effect
RESET	1	I	_	RESET input pin
RESETLOC(1,0)	2	I	_	No effect
ROMEN	1	I	_	No effect
LD(31-0)	32	I/O/Z	S	Set to high-impedance
TCLK0	1	I/O	Α	Set to high-impedance
TCLK1	1	I/O	А	Set to high-impedance

[†]A = Asynchronous, S = Synchronous

[‡] Recommended decoupling capacitors are one multiple 0.1 μ F and 4.7 μ F around the device. Number depends on specific board noise conditions.

(I) Power (70 pins)								
Signal	Pins	I∕O§	Type [†]	Description				
SUBS	1	I	_	Substrate pin (tie to ground). Set to high-impedance.				
V _{SSL}	4	Ι	_	Ground pins. Set to high-impedance.				
CV _{SS}	15	Ι	_	Ground pins. Set to high-impedance.				
DV_{SS}	15	Ι	_	Ground pins. Set to high-impedance.				
IV _{SS}	6	Ι	_	Ground pins. Set to high-impedance.				
DV _{DD}	13	Ι	_	+5V _{DC} supply pins. Set to high-imped- ance.‡				
GADV _{DD}	3	Ι	_	+5V _{DC} supply pins. Set to high-imped- ance.‡				
GDDV _{DD}	3	Ι	_	+5V _{DC} supply pins. Set to high-imped- ance.‡				
LADV _{DD}	3	Ι	_	+5V _{DC} supply pins. Set high-imped- ance. [‡]				
LDDV _{DD}	3	Ι	_	+5V _{DC} supply pins. Set to high-imped- ance. [‡]				
V _{DDL}	4	I	_	+5V _{DC} supply pins. Set to high-imped- ance. [‡]				

Table 7–3. Pin States After System Reset (Continued)

[†] A = Asynchronous, S = Synchronous [‡] Recommended decoupling capacitors are one multiple 0.1 μ F and 4.7 μ F around the device. Number depends on specific board noise conditions.

When RESET is released, the 'C4x begins executing the application program. The initial address of the program is stored in the reset vector. The 'C4x permits selection of any one of four reset vector locations. Selection of the reset vector location that is used is determined by the levels on the RESETLOC1 and RESETLOC0 pins at reset. Table 7–4 shows the possible configurations of these pins.

Table 7-4. RESET Vector Locations

Value at RES	ETLOCx Pin	Get Reset Vector From	
RESETLOC1	RESETLOC0	Hex Memory Address	Comment
0	0	00000 0000	Local Bus
0	1	07FFF FFFF [†]	Local Bus
1	0	08000 0000†	Global Bus
1	1	OFFFF FFFF [†]	Global Bus

[†] This corresponds to the 32-bit address that the processor accesses. However, in the 'C44 only the 24-LSBs of the reset address will be driven on pins A0–A23 or pins LA0–LA23. The corresponding LSTRBx pins will also be activated.

7.7.3 Additional Reset Operations

After system reset (after RESET goes back from 0 to 1), the following additional operations are performed:

- Timer registers are set.
 - The timer global control register is set to 0, except that bit DATIN is set to the value on pin TCLK.
 - The timer counter and timer period registers set to zeros.
- □ Control registers for communication ports 0–2 (subsection 12.3.1 on page 12-8) are set to zeros (output operation), and control registers for communication ports 3–5 are set to 04h (input operation).
- External memory interface control registers (Section 9.3 on page 9-6) are set to 3E39 FFF0h. (7 wait states)
- DMA channel control register, DMA transfer counter, and DMA auxiliary transfer counter (subsection 11.3.1 on page 11-7) are set to zeros.

- The following CPU registers are loaded with zeros (each described in Chapter 3):
 - IIE (CPU internal interrupt enable register)
 - IIF (interrupt flag register)
 - DIE (DMA internal enable register)
 - IVTP (interrupt-vector table pointer)
 - TVTP (trap-vector table pointer)
- ☐ The CPU status register (ST) is set to 0400h, which puts the on-chip cache in *cache freeze* mode.
- The reset vector is read from its location and loaded into the PC.
- ☐ If ROMEN=1 (Internal ROM enabled), the RESETLOC(1,0) pins are low, and the IIOF0 pin is high, the 'C4x will start execution of the bootloader code. Otherwise, the 'C4x will start execution of the routine which is pointed to by the reset vector corresponding to the RESETLOC(1,0) pins.

Multiple 'C4xs driven by the same system clock can be reset and synchronized. See *Reset Signal Generation* in the *TMS320C4x General-Purpose User's Guide* for information about resetting multiple 'C4xs.

Chapter 8

Pipeline Operation

Two characteristics of the 'C4x that contribute to its high performance are pipelining and concurrent I/O and CPU operation.

Four functional units control 'C4x pipeline operation: fetch, decode, read, and execute. Pipelining is the overlapping or parallel operations of the fetch, decode, read, and execute levels of a basic instruction.

The DMA coprocessor decreases pipeline interference and enhances the CPU's computational throughput by performing input/output operations.

c Page
Pipeline Structure 8-2
Pipeline Conflicts
Memory Accesses for Maximum Performance
Clocking of Memory Accesses

8.1 Pipeline Structure

The four major units of the 'C4x pipeline structure and their functions are as follows:

Fetch Unit (F)	Fetches the instruction words from memory and updates the program counter (PC).
Decode Unit (D)	Decodes the instruction word and performs ad- dress generation. Also, controls modification of the ARn registers in the indirect addressing mode, and of the stack pointer when PUSH to/ POP from the stack occurs.
Read Unit (R)	If required, reads the operands from memory.
Execute Unit (E)	If required, reads the operands from the regis- ter file, performs the necessary operation, and writes results to the register file. If required, re- sults of previous operations are written to memory.

A basic instruction has four levels: fetch, decode, read, and execute. Figure 8–1 illustrates these four levels of the pipeline structure. The levels are indexed according to instruction and execution cycle. In the figure, perfect overlap in the pipeline, where all four units operate in parallel, occurs at cycle (*m*). Levels about to be executed are at m+1, and those just executed are at m-1. The 'C4x pipeline controller supports a high-speed processing rate of one execution per cycle. It also manages pipeline conflicts so that they are transparent to the user. You do not need to take any special precautions to guarantee correct operation.

Figure 8–1. Pipeline Structure

CYCLE	Fetch	Decode	Read	Execute		
m–3	W	_	_	_		
m–2	Х	W	_	_		
m–1	Y	Х	W	_		
m	Ζ	Y	Х	W		Perfect overlap
m+1	_	Ζ	Y	Х		
m+2	_	_	Ζ	Y		
m+3	_	_	_	Z		
Notes:	1) W, X, Y, a	and Z repres	ent instruc	tions.		
	2) F, D, R, E	E = fetch, deo	code, read,	and execute,	respe	ctively.

Priorities from highest to lowest have been assigned to each of the functional units of the pipeline and to the DMA controller as follows:

- DMA (if configured as highest priority)
- Execute
- Read
- Decode
- Fetch
- DMA (if configured as lowest priority).

When the processing of an instruction is ready to pass to the next higher pipeline level and that level is not ready to accept a new input, a pipeline conflict occurs. In this case, the lower priority unit waits until the higher priority unit completes its currently executing function.

8.2 Pipeline Conflicts

Pipeline conflicts in the 'C4x can be grouped into the following three main categories:

Branch Conflicts	Involve most of those instructions or operations that read and/or modify the PC.
Register Conflicts	Involve delays that can occur when reading from or writ- ing to registers that are used for address generation, such as: AR0–AR7, IR0, IR1, BK, DP and SP.
Memory Conflicts	Occur when the internal units of the 'C4x compete for memory resources.

Each of these three types is discussed in the following subsections. Examples are included. Note in these examples, when data is refetched or an operation is repeated, the symbol representing the stage of the pipeline is appended with a number. For example, if a fetch is performed again, the instruction mnemonic is repeated. The symbol $\overline{\text{RDY}}$ is used to indicate that a unit is not ready and the symbol RDY is used to indicate that a unit is ready.

8.2.1 Branch Conflicts

Branch conflicts involve most of the instructions or operations that read and/or modify the PC.

8.2.1.1 Standard Branches

Pipeline conflicts occur with standard (nondelayed) branches, i.e., BR, B*cond*, DB*cond*, CALL, IDLE, RPTB, RPTS, RETI*cond*, RETS*cond*, interrupts, and reset, because their execution is all the pipeline can handle. Other information fetched into the pipeline is discarded or refetched, or the pipeline becomes inactive; this is referred to as flushing the pipeline. Flushing the pipeline is necessary in these cases to prevent partial execution of succeeding instructions. The branches discussed here are loads; TRAP *cond* and CALL*cond* are treated as conditional stores and are shown in Example 8–13.

Example 8–1 shows the code and pipeline operation for a standard branch. Note that one dummy fetch is performed (MPYF instruction), and then after the branch address is available, a new fetch (OR instruction) is performed. This dummy fetch introduces the MPYF instruction into the cache.

Example 8–1. Standard Branch

В	R THREE MPYF ADD SUBF AND	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;		Unconditional branch Not executed Not executed Not executed Not executed
	THREE TI	OR ;	1	Fetched after BR is taken



PIPELINE OPERATION

Note:

Both RPTS and RPTB flush the pipeline, allowing the RS, RE, and RC registers to be loaded at the proper time. If these registers are loaded without the use of RPTS or RPTB, no flushing of the pipeline occurs. Thus, RS, RE, and RC can be used as general-purpose 32-bit registers without pipeline conflicts. When RPTB is nested because of nested interrupts, it may be necessary to load and store these registers directly while using the repeat modes. Since up to four instructions can be fetched before the repeat mode is entered, loads should be followed by a branch to flush the pipeline. If the RC is changing when an instruction is loading it, the direct load takes priority over the modification made by the repeat mode logic.

8.2.1.2 Delayed Branches Without Annul Option

Delayed branches are implemented to assure that the next three instructions are fetched and executed. The delayed branches without annul option include BRD, B*cond*D, and DB*cond*D. Example 8–2 shows the code and pipeline operation for a delayed branch.

Example 8–2. Delayed Branch Without Annul Option

BRD THREE MPYF ADD SUBF AND	;;;;;;	Unconditional Executed Executed Not executed	delayed	branch
THREE MPYF	;	Fetched after	SUBF is	fetched

PIPELINE OPERATION

PC	Fetch	Decode	Read	Execute	
n	BRD	-	_	-	
n+1	MPYF	BRD	_	_	No execute delay
n+2	ADDF	MPYF	BRD	-	
n+3	SUBF	ADDF	MPYF	BRD	– THREE $ ightarrow$ PC
THREE	MPYF	SUBF	ADDF	MPYF	

8.2.1.3 Delayed Branches With Annul Option

The 'C4x supports delayed branches with an annulling option: BcondAT (branch conditional, annul if true) and BcondAF(branch conditional, annul if false). The true or false status of the condition controls whether or not a branch is performed (as in a delayed branch). The annulling operation cancels the effect of the execute phase of the first instruction and of the read and execute phases of the second and third instructions following the BcondAT or BcondAF.

- □ If the condition is true, B*cond*AT performs a branch, and the annulling operation takes place. Otherwise, the branch is not taken and the annulling operation does not take place.
- □ If the condition is false, B*cond*AF does not perform a branch, and the annulling operation takes place. Otherwise, the branch is taken and the annulling operation does not take place.

See subsection 7.2.2 for more information about delayed branches with annulling. Example 8–3 uses both B*cond*AT and B*cond*AF.

Example 8–3. Using BcondAF and BcondAT Instructions

top:	LDI BNAT ADDI MPYF NOT SUBF	*AR1,R0 bottom *++AR2,R3	;;;;;;	If negative, branch and annul the execute phase of ADDI, MPYF, and NOT. Otherwise, don't annul and continue with SUBF.
bottor	SUBI BNNAF ADDI MPYF NOT m:XOR	1,R0 top *++AR2,R3	;;;;;;;	If not negative, branch and do not annul the execute phase of ADDI, MPYF, and NOT. Otherwise, annul ADDI, MPYF, and NOT, and continue with XOR.

At the start of Example 8–3, if the result of the load is *negative* (a *true* condition), the B*cond*AT instruction causes a branch and also annuls the execute phase of the three instructions that follow it. As a result, the execute phase of the ADDI instruction does not occur, and register R3 is not updated by addition. However, AR2 is incremented, and data at the corresponding address is read because these operations are in the decode and read phases of the pipeline, respectively, and thus cannot be annulled.

Pipeline Operation 8-7

Two types of operations can be annulled:

- □ All writes to the register file that occur in the execute phase (ADDs, LDs, etc., but not LDA, LDPK, etc.)
- □ All stores to memory

8.2.2 Register Conflicts

A register conflict occurs if you read from or write to a register used for addressing purposes (AR0–AR7, IR0, IR1, BK, DP, and SP) when the register is not ready to be used. For example, if an instruction writes to one of these registers, the decode unit cannot use that same register until the write is complete (which occurs in the execute stage).

In Example 8–4, an auxiliary register is loaded, and the same auxiliary register is used on the next instruction. Since the decode stage needs the result of the write to the auxiliary register, the decode of this second instruction is delayed two cycles. Every time the decode is delayed, a refetch of the program word is performed; i.e., ADDF is fetched three times. Because these are actual refetches, they can cause not only conflicts with the DMA controller, but also cache hits and misses. If the AR register used in the MPYF instruction were different from the one used in the LDI instruction, no delay would occur.

Example 8–4. Write to an AR Followed by an AR for Address Generation





PIPELINE OPERATION

Conflicts involving reads are similar to those involving writes. If an instruction must read registers AR0–AR7 or SP, the use of those particular registers by the decode stage for the following instruction is delayed until the read is complete. The registers are read at the start of the execute cycle and therefore require only a one-cycle delay of the following decode. For four registers (IR0, IR1, BK, or DP), no delay is incurred upon a read.

In Example 8–5, two auxiliary registers are added together with the result going to an extended-precision register. The next instruction uses one of the same auxiliary registers as an address register. If the MPYF instruction used an AR register other than AR0 or AR2, *no delay would occur*.

Example 8–5.A Read of ARs Followed by ARs for Address Generation





Note:

The DBR (decrement and branch) instruction's use of auxiliary registers for loop counters is treated the same as if the use were for addressing. Therefore, the operation shown in the two previous examples can also occur for this instruction.

8.2.3 Memory Conflicts

Memory conflicts can occur when the memory bandwidth of a physical memory space is exceeded. RAM blocks 0 and 1 and the ROM block can support only two accesses per cycle. The external interface can support only one access per cycle. Some conditions under which memory conflicts can be avoided are discussed in Section 8.3, on page 8-17.

Memory pipeline conflicts consist of the following four types:

A program fetch is prevented from beginning.
A program fetch has begun but is not yet complete.
An instruction sequence requires three CPU data accesses in a single cycle.
A global or local bus operation must complete before another one can proceed.

These four types of memory conflicts are illustrated in examples and discussed in the paragraphs that follow.

8.2.3.1 Program Wait

Two conditions can delay an instruction fetch:

- Too many accesses to the same memory at the start of a CPU data access can occur in two cases:
 - Two CPU data accesses are made to an internal RAM or ROM block, and a program fetch from the same block is necessary.
 - One of the external ports is starting a CPU data access, and a program fetch from the same port is necessary.
- A multicycle CPU data access or DMA data access over the external bus is needed.

Example 8–6 illustrates a program wait until a CPU data access completes. In this case, *AR0 and *AR1 are both pointing to data in RAM block 0, and the MPYF instruction will be fetched from RAM block 0. This results in the conflict shown. Since no more than two accesses can be made to RAM block 0 in a single cycle, the program fetch cannot begin and must wait until the CPU data accesses are complete.

Example 8–6. Program Wait Until CPU Data Access Completes

ADDF3 *AR0,*AR1,R0 FIX MPYF ADDF3 NEGB



PIPELINE OPERATION

Example 8–7 shows a program wait due to a multicycle data-data access or a multicycle DMA access. The ADDF, MPYF, and SUBF are fetched from some portion in memory other than the external port the DMA requires. The DMA begins a multicycle access. The program fetch corresponding to the CALL is made to the same external port that the DMA is using.

Even if the DMA is configured as the lowest priority, **a multicycle access cannot be aborted**. The program fetch must therefore wait until the DMA access completes.

PC	I	Fetch	Decode	I	Read	I	Execute	
n		ADDF	_		-		-	
n+1		MPYF	ADDF		_		_	
n+2		SUBF	MPYF		ADDF		_	
n+3		(wait)	SUBF		MPYF		ADDF	
n+3		CALL	(nop)		SUBF		MPYF	
n+4		_	CALL		(nop)		SUBF	

PIPELINE OPERATION

Example 8–7. Program Wait Due to Multicycle Access

8.2.3.2 Program Fetch Incomplete

A program fetch incomplete occurs when an instruction fetch takes more than one cycle to complete because of wait states. In Example 8-8, the MPYF and ADDF are fetched from memory that supports single-cycle accesses. The SUBF is fetched from memory requiring one wait state. One example that demonstrates this conflict is a fetch across a bank boundary on the external port.

Example 8–8. Multicycle Program Memory Fetches

PIPELINE OPERATION Fetch Decode Execute Read



8.2.3.3 Execute Only

The Execute Only type of memory pipeline conflict occurs when a sequence of instructions requires three CPU data accesses in a single cycle. There are two cases in which this occurs:

- An instruction performs a store and is followed by an instruction that performs two memory reads.
- ☐ An instruction performs two stores and is followed by an instruction that performs at least one memory read.

The first case is shown in Example 8–9. Since this sequence requires three data memory accesses and only two are available, only the execute phase of the pipeline is allowed to proceed. The dual reads required by the LDF || LDF are delayed one cycle. Note that in this case a refetch of the next instruction can occur, which could cause an additional access to memory.

Example 8–9. Single Store Followed by Two Reads

STFR0,*AR1 ; R0 \rightarrow *AR1 LDF*AR2,R1 ; *AR2 \rightarrow R1 in parallel with || LDF*AR3,R2 ; *AR3 \rightarrow R2

PC	Fetch	Decode	Read	Execute	
n	STF	-	-	_	
n+1	ldf ldf	STF	-	_	
n+2	W	ldf ldf	STF	_	Write must
n+3	Х	W	ldf ldf	STF	before the 2 reads can
n+4	Х	W	ldf ldf	(nop)	complete.
n+4	Y	Х	W	ldf ldf	

PIPELINE OPERATION
Example 8–10 shows a parallel store followed by a single load or read. Since two parallel stores are required, the next CPU data memory read must wait one cycle before beginning. One program memory refetch may occur.

Example 8–10. Parallel Store Followed by Single Read

STF R0,*AR0 ; R0 \rightarrow *AR0 in parallel with || STF R2,*AR1 ; R2 \rightarrow *AR1 ADDF @SUM,R1 ; R1 + @SUM \rightarrow R1 IACK ASH



PIPELINE OPERATION

8.2.3.4 Hold Everything

Three types of conditions cause Hold Everything memory pipeline conflicts:

- A CPU data load or store cannot be performed because an external port is busy.
- An external load takes more than one cycle.
- ☐ The execution of conditional calls and traps, which take one more cycle than conditional branches.

The first type of Hold Everything conflict occurs when one of the external ports is busy because of an access that has started but is not complete. In Example 8–11, the first store is a two-cycle store. The CPU writes the data to an external port. The port control then takes two cycles to complete the datadata write. The LDF is a read over the same external port. Since the store is not complete, the CPU continues to attempt processing the LDF until the port is available.

Example 8–11. Busy External Port

STF	R0,@DMA1
LDF	@DMA2,R0

PC	Fetch	Decode	Read	Execute	
n	STF	_	_	_	
n+1	LDF	STF	_	_	
n+2	W	LDF	STF	_	
n+2	W	LDF	(nop)	STF	
n+2	W	LDF	(nop)	(nop)	2-cycle external bus ↓ write access
n+3	Х	W	LDF	(nop)	
n+4	Y	Х	W	LDF	

PIPELINE OPERATION

The second type of Hold Everything conflict involves multicycle data reads. In this case, the read has begun and continues until completed. In Example 8–12, the LDF is performed from an external memory that requires several cycles to access.

	LDF	@DMA,RC)		
	PIPE	LINE OPERA	TION		
PC	Fetch	Decode	Read	Execute	1
n	LDF	-	_	_	
n+1	I	LDF	_	_	
n+2	J	I	LDF	_	▲ 2-cycle external bus
n+3	K(dummy)	I	LDF	-	read access ▼
n+3	K ₂	J	I	LDF	

Example 8–12. Multicycle Data Reads

The final type of Hold Everything conflict deals with conditional calls (CALL*cond*) and traps (TRAP*cond*), which are different from other branch instructions. Whereas other branch instructions are conditional loads, the conditional calls and traps are conditional stores, which take one more cycle to complete than conditional branches (see Example 8–13). The added cycle pushes the return address after the call condition is evaluated.

Example 8–13. Conditional Calls and Traps

		-	-		
PC	Fetch	Decode	Read	Execute	I
n	CALLcond	_	_	_	
n+1	I	CALLcond	-	-	
n+1	(nop)	(nop)	CALLcond	-	
n+1	(nop)	(nop)	(nop)	CALLcond	PC store
n+1	(nop)	(nop)	(nop)	CALLcond	cycle
n+2/CALLaddr	I	(ndp)	(nop)	(nop)	

PIPELINE OPERATION

8.3 Memory Accesses for Maximum Performance

If program fetches and data accesses are performed in such a manner that the resources being used cannot provide the necessary bandwidth, the pipeline is stalled until the accesses are complete. Certain configurations of program fetch and data accesses yield conditions under which the 'C4x can achieve maximum throughput.

Table 8–1 shows how many accesses can be performed from the different memory spaces when it is necessary to do a program fetch and a single data access, and still achieve maximum performance (one cycle). Four cases achieve one-cycle maximization.

Table 8–1. One Program Fetch and One Data Access for Maximum Performance

Case No.	Global Bus Accesses	Accesses From Dual-Access Internal Memory	Local Bus Or Peripheral Accesses
1	1	1	—
2	1	—	1
3	—	2 from any combination of internal memory	_
4	—	1	1

Table 8–2 shows how many accesses can be performed from the different memory spaces when it is necessary to do a program fetch and two data accesses, still achieving maximum performance (one cycle). Six cases achieve this maximization.

Case No.	Global Bus Accesses	Accesses From Dual-Access Internal Memory	Local Or Peripheral Bus Accesses
1	1	2 from any combination of internal memory	_
2†	1 program	1 data	1 data
34	1 data	1 data	1 program
4	1 data	1 program, 1 data	1 DMA
5	_	2 from same internal memory block and1 from a different internal memory block	_
6	—	3 from different internal memory blocks	1 DMA
7	_	2 from any combination of internal memory	1
8	1 program	2 data	1 DMA
9	1 DMA	2 data	1 program

Table 8–2. One Program Fetch and Two Data Accesses for Maximum Performance

[†] For Cases 2 and 3, see Three-Operand Instruction Memory Reads on page 8-20.

8.4 Clocking of Memory Accesses

This section discusses the role of internal clock phases (H1 and H3) in the way the 'C4x handles multiple memory accesses. Whereas the previous section discussed the interaction between sequences of instructions, this section discusses the flow of data on an individual instruction basis.

Each major clock period of 40 ns is composed of two minor clock periods of 20 ns, labeled H3 and H1 (these times assume a 50-MHz 'C40). The active clock period for H3 and H1 is the time when that signal is high.



The precise operation of memory reads and writes can be defined according to these minor clock periods. The types of memory operations that can occur are program fetches, data loads and stores, and DMA accesses. Internal DMA data accesses always start during the H3 cycle.

8.4.1 Program Fetches

Internal program fetches are always performed during H3 unless a single data store must occur at the same time because of another instruction in the pipeline. In that case, the program fetch occurs during H1 and the data store occurs during H3.

External program fetches always start at the beginning of H3 with the address being presented on the external bus. At the end of H1, the fetches are completed with the latching of the instruction word.

8.4.2 Data Loads and Stores

Four types of instructions perform loads, memory reads, and stores: two-operand instructions, three-operand instructions, multiplier/ALU operation with store instructions, and parallel multiply and add instructions. See Chapter 6 for detailed information on addressing modes.

As discussed in Chapter 9, the number of bus cycles for external memory accesses differs in some cases from the number of CPU execution cycles. For external reads, the number of bus cycles and CPU execution cycles is identical. For external writes, there are always at least two bus cycles, but unless there is a port access conflict, there is only one CPU execution cycle. In the following examples, any difference in the number of bus cycles and CPU cycles is noted.

8.4.2.1 Two-Operand Instruction Memory Accesses

Figure 8–2. Two-Operand Instruction Word

31	24 23	}	16	15 8	7 0
0 X 0	Operation	G	dst(src)	sro	c(dst)

Two-operand instructions include all those instructions with bits 31-29 being 000_2 or 010_2 (see Figure 8–2). In the case of a data read, bits 15–0 represent the *src* operand. Internal data reads are always performed during H1. External data reads always start at the beginning of H3 with the address presented on the external bus, and they complete with the latching of the data word at the end of H1.

In the case of a data store, bits 15–0 represent the *dst* operand. Internal data stores are performed during H3. External data stores always start at the beginning of H3 with the address and data presented on the external bus.

8.4.2.2 Three-Operand Instruction Memory Reads

Figure 8–3. Three-Operand Instruction Word

31	2423	8	16	15	8 7 0		
001	Operation	Т	dst	src1		src2	

Three-operand instructions include all instructions with bits 31-29 being 001_2 (see Figure 8-3). The source operands, *src1* and *src2*, come from either registers or memory. When one or more of the source operands are from memory, these instructions are always memory reads.

If only one of the source operands is from memory (either *src1* or *src2*) and is located in internal memory, the data is read during H1. If the single memory

source operand is in external memory, the read starts at the beginning of H3, with the address presented on the external bus, and completes with the latching of the data word at the end of H1.

If both source operands are to be fetched from memory, then memory reads can occur in several ways:

- If both operands are located in internal memory, the *src1* read is performed during H3 and the *src2* read during H1, thus completing two memory reads in a single cycle.
- ☐ If *src1* is in internal memory and *src2* is in external memory, the *src2* access begins at the start of H3 and latches at the end of H1. At the same time, the *src1* access to internal memory is performed during H3. Again, two memory reads are completed in a single cycle.
- If src1 is in external memory and src2 is in internal memory, two cycles are necessary to complete the two reads. In the first cycle, the internal src2 access is performed. The src1 is also performed, but not latched until the next H3.
- ☐ If *src1* and *src2* are both from external memory, two cycles are required to complete the two reads. In the first cycle, the *src1* access is performed and loaded on the next H3; in the second cycle, the *src2* access is performed and loaded on that cycle's H1.

8.4.2.3 Operations with Parallel Stores

Figure 8–4. Multiply or CPU Operation With a Parallel Store

31		24 23		16	15	87	0
1 1	Operation	dst1	src1	src3	dst2		src2

The next class of instructions includes all instructions that have a store in parallel with another instruction. Bits 31 and 30 for these instructions are equal to 11_2 .

For operations that perform a multiply or ALU operation in parallel with a store, the instruction word format is shown in Figure 8–4. If the store operation to *dst2* is external or internal, it is performed during H3. Two bus cycles are required for external stores, but only one CPU cycle is necessary to complete the write.

If the memory read operation is external, it starts at the beginning of H3 and latches at the end of H1. If the memory read operation is internal, it is performed during H1. Note that memory reads are performed by the CPU during the read (R) phase of the pipeline, and stores are performed during the execute (E) phase.

The instruction word format for instructions that have parallel stores to memory is shown in Figure 8–5. If both destination operands, *dst1* and *dst2*, are located in internal memory, *dst1* is stored during H3 and *dst2* during H1, thus completing two memory stores in a single cycle.

Figure 8–5. Two Parallel Stores

31	24 23				16	15 8	8 7		
1 1	ST ST	src2	0	0 0	src1	dst1	dst2		

If *dst1* is in external memory and *dst2* is in internal memory, the *dst1* store begins at the start of H3. The *dst2* store to internal memory is performed during H1. Two bus cycles are required for the external store, but only one CPU cycle is necessary to complete the write. Again, two memory stores are completed in a single cycle.

If *dst1* is in internal memory and *dst2* is in external memory, an additional bus cycle is necessary to complete the *dst2* store. Only one CPU cycle is necessary to complete the write, but the port access requires three bus cycles. In the first cycle, the internal *dst1* store is performed during H3, and *dst2* is written to the port during H1. During the next cycle, the *dst2* store is performed on the external bus, beginning in H3, and executes as normal through the following cycle.

If *dst1* and *dst2* are both written to external memory, a single CPU cycle is still all that is necessary to complete the stores. In this case, four bus cycles are required.

- 1) In the first cycle, both *dst1* and *dst2* are written to the port, and the external bus access for *dst1* begins.
- 2) The store for *dst1* is completed on the second cycle.
- 3) The store for *dst2* begins on the third external bus cycle.
- 4) Finally, the store for *dst2* is completed on the fourth external bus cycle.

8.4.2.4 Parallel Multiplies and Adds

Memory addressing for parallel multiplies and adds is similar to that for threeoperand instructions. The parallel multiplies and adds include all instructions with bits 31-30 equal to 10_2 (see Figure 8–6).

Figure 8-6. Parallel Multiplies and Adds

31	2423				16	15	8	7 0
1 0	Operation	Ρ	d1 d	2 src1	src2	src3		src4

For these operations, *src3* and *src4* are both located in memory. If both operands are located in internal memory, *src3* is performed during H3, and *src4* is performed during H1, thus completing two memory reads in a single cycle.

If *src3* is in internal memory and *src4* is in external memory, the *src4* access begins at the start of H3 and latches at the end of H1. At the same time, the *src3* access to internal memory is performed during H3. Again, two memory reads are completed in a single cycle.

If *src3* is in external memory and *src4* is in internal memory, two cycles are necessary to complete the two reads. In the first cycle, the internal *src4* access is performed. During the H3 of the next cycle, the *src3* access is performed.

If *src3* and *src4* are both from external memory, two cycles are necessary to complete the two reads. In the first cycle, the *src3* access is performed; in the second cycle, the *src4* access is performed.

Chapter 9

External Bus Operation

The 'C4x has two identical external bus interfaces. One bus is called the global memory interface and the other bus is called the local memory interface. These buses are designed to allow higher throughput by permitting simultaneous loads and stores to different external memories.

The information in this chapter applies to both the global memory interface and the local memory interface; however, in some sections, only the global memory interface is shown. Examples of memory interfacing are provided in the *TMS320C4x General-Purpose Applications User's Guide*.

Topic

Page

9.1	Overview
9.2	Memory Interface Signals 9-3
9.3	Memory Interface Control Registers
9.4	Programmable Wait States 9-14
9.5	Memory Interface Timing
9.6	Using Enable Signals to Control Signal Groups
9.7	Interlocked Operations9-39
9.8	IACK Timing

9.1 Overview

The 'C4x has two identical parallel external interfaces: the *global memory interface* and the *local memory interface*. Each interface has the following features:

- Separate configurations, each with its own 32-bit data bus and 31-bit address bus (24 pin address bus in the 'C44)
- Single-cycle reads and pipelined writes
- Independent enable signals for data, address, and control lines
- Bus-request and bus-lock signaling for shared memory parallel processing
- User-controlled mapping of addresses to either of two sets of independent strobes for different speed memories
- Look-ahead bus status signals for defining current and requested bus operations for parallel processing arbitration
- Selectable wait states (both software- and hardware-controlled)
- Signals that indicate when memory-page boundaries are crossed.

Note:

The global-memory interface is identical in every way to the local memory interface except that (1) they have different positions in the memory map, and (2) the control signals for the local memory interface are labeled an additional "L" prefix (as described in Figure 9–1 on page 9-3).

Throughout this chapter, no distinction is made between global and local interface signals and between STRB0 and STRB1, except for clarity.

The signals that indicate when memory-page boundaries are crossed support three main types of memory:

- page-mode and static-column decode DRAMs
- high-speed SRAM banks
- slow speed memory banks and I/O devices

9.2 Memory Interface Signals

As shown in Figure 9–1, the global-memory interface has two sets of control signals, STRB0 and STRB1. The global-memory port control-registers (Section 9.3 on page 9-6) define which set of registers is active.

Figure 9–1. Global and Local Memory Interface Control Signals



Note: The signals used in this figure are for the global-memory interface. The local-memory interface signals have the same configuration and an additional "L" prefix is added for each signal (for example, STRB0 becomes LSTRB0, etc.).

Signal†	Type §	Description	Value After Reset	ldle Status [∥]
AE¶	I	Address bus enable signal for global-memory interface. When high (set to 1), places address lines A30–0 in the high- impedance state.	N.A.#	ignored
<u>CE</u> (0,1)¶	I	Control signal enable for R/Wx , \overline{STRBx} , and $PAGEx$ signals. When high (set to 1), it places the corresponding R/Wx , \overline{STRBx} , and $PAGEx$ signals in high-impedance state (x = 0 for $\overline{CE0}$ and x = 1 for $\overline{CE1}$).	N.A.	ignored
DE¶	I	Data bus enable signal for global memory interface. When high (set to 1), places data lines D31-0 in the high-imped- ance state. Reads can still occur but writes cannot.	N.A.	ignored
LOCK‡	0	Lock signal for global bus interface. Indicates whether an interlocked access is underway ($0 = access$ underway; $1 = access$ not underway). LOCK is changed <i>only</i> by the interlocked instructions.	1	1
PAGE(0,1)	O/Z	Memory-page enable signal for STRB(0,1) accesses	0	0
RDY(0,1)	I	Indicates external memory is ready to be accessed	N.A.	ignored
R/W(0,1)	O/Z	Specifies memory read (active high) or write (active low) mode	1	1
STAT(3-0) ‡	0	Four lines that define the status or function of the memory port as shown in Table 9–2 (next page).	all 1s	all 1s
STRB(0,1)	O/Z	Interface access strobe	1	1
A(30–0)	O/Z	Address bus. The address lines are always driven. They keep the address of the last access.	Hi–Z	address of last access
D(31–0)	I/O/Z	Data bus. These signals go to high impedance between write accesses.	Hi–Z	Hi–Z

Table 9–1. Global Memory Interface Signals

[†] The numbers in parentheses mean that either a 0 (zero) or a 1 can follow the prefix shown to the left of the parenthesis. A zero indicates STRB0 control signals (shown in Figure 9–1), and a one indicates STRB1 control signals.

[‡]STAT(3-0) and LOCK cannot be controlled by an external control signal.

§ O=output; I=input; Z=high-impedance state.

This signal can be used in a shared bus configuration to hold the 'C4x off the shared bus while another 'C4x accesses the shared memory and peripherals.

#N.A. means not affected.

|| Idle status = no external memory access

Table 9–2 shows how pins STAT3 to STAT0 define the current status of the global-memory port. For bus accesses, these signals provide information about the access that is about to begin. The code for a SIGI instruction read is useful for distinguishing between a SIGI read and a LDII or LDFI read.

The bus idle status code is 1111_2 (given at the bottom of Table 9–2). This simplifies modular shared-bus multiprocessor interfaces because pull-up resistors can be used to signal the idle condition when processor cards are not attached to the shared bus.

Table 9–2. Global Memory Port Status for STRB0 and STRB1 Accesses

	Value a	t Pins †		
STAT3	STAT2	STAT1	STAT0	Status
0	0	0	0	STRB0 access, program read
0	0	0	1	STRB0 access, data read
0	0	1	0	STRB0 access, DMA read
0	0	1	1	STRB0 access, SIGI (instruction) read
0	1	0	0	Reserved
0	1	0	1	STRB0 access, data write
0	1	1	0	STRB0 access, DMA write
0	1	1	1	Reserved
1	0	0	0	STRB1 access, program read
1	0	0	1	STRB1 access, data read
1	0	1	0	STRB1 access, DMA read
1	0	1	1	STRB1 access, SIGI (instruction) read
1	1	0	0	Reserved
1	1	0	1	STRB1 access, data write
1	1	1	0	STRB1 access, DMA write
1	1	1	1	ldle

[†] This table applies to both the global-memory interface and local-memory interface (for local memory interface signals, add an L prefix to form LSTAT3, LSTAT2, etc.).

9.3 Memory-Interface Control Registers

Figure 9–2 shows the memory map for both the global- and local-memory interface-control registers. Figure 9–3 shows the fields in each register. Each register can be programmed to control its respective memory interface by defining the:

- Page size used for the two strobes at each port
- Address ranges over which the strobes are active
- Wait states
- Other operations that control the memory interface

Figure 9–3 lists the fields in these registers.

At reset, the binary values shown above each bit in Figure 9–2 are written to the global memory interface control register. Values in bits 3-0 are the values at these bits' respective pins (\overline{AE} , \overline{DE} , $\overline{CE1}$, and $\overline{CE0}$). Reset has the following effects (for both the local bus and the global bus):

- ☐ The PAGESIZE fields for STRB0 (bits 18–14) and STRB1 (bits 23–19) are set to 00111₂, which corresponds to 256 words.
- ☐ The WTCNT fields for STRB0 (bits 10–8) and STRB1 (bits 13–11) are set to 111₂, which corresponds to seven wait states.
- The ACTIVE field for STRB0 (bits 28–24) is set for all addresses over the global (or local for LSTRB0) memory interface.
- The STRB SWITCH field (bit 29) is set to 1 to insert a cycle between backto-back reads that switch from STRB0 to STRB1 (or STRB1 to STRB0).
- The SWW fields for STRB0 (bits 5–4) and STRB1 (bits 7–6) are both set to 11₂ to set the internal ready signal to be the logical AND of the external READY signal (RDY) and the ready signal generated by the on-chip waitstate counter (RDY_{wtcnt}).

Figure 9–2. Location of the Memory-Interface Control Registers

00010 0000h	Global memory interface control register
00010 0001h	
	Reserved
00010 0003h	
00010 0004h	Local memory interface control register

Figure 9–3.	Fields in	the Memor	y-Interface	Control	Registers
			/		

					_	31	30	29	28	3				24	23					19
XX XX STRB SWITCH							(Ta	STRE ble 9-	3 ACTI -4, Tab	VE le 9–5))		STF	RB1 PA (Table	AGES 9–3)	IZE				
					-	R 0	R 0	RW 1	RV 1	VR	W 1	RW 1	RW 1	RW 0	RW 0	RW 0	RW 1	/ F	RW 1	RW 1
_	18				14	13		11	10		8	7	6	6	5	4	3	2	1	0
l		STRB (T	0 PAG able 9-	iesize —3)	1	STR	B1 W1	CNT	STRI	B0 W ⁻	TCNT	STR (Ta	B1 SV ble 9–4	VW 6)	STRB0 (Table	SWW 9–6)	AE	DE	CE1	CE0
	RW	RW 0	RW 1	RW 1	RW 1	RW 1	RW 1	RW 1	RW 1	RW 1	RW 1	RW 1	R	W 1	RW 1	RW 1	R	R	R	R

Notes: 1) The register cell figure contains global-memory interface-control register mnemonics. For local-memory interfacecontrol register mnemonics, add an L prefix to each mnemonic in the figure (e.g., LSTRB SWW, LCE0, etc.).

2) The 1s and 0s below each bit are the binary values written to the register at reset. The values at bits 3–0 are defined by the values of their respective external pins (AE, DE, CE1, and CE0).

3) These registers are shown in the overall memory map in Figure 4–1 and Figure 4–3.

4) RW=read/write; R=read.

Note:

Mnemonics used are for the global memory interface control register. For the local-memory interface-control register, add the prefix L to each mnemonic (e.g., LCEO, LCE1, LSTRB1, etc. The description remains the same for the local-memory interface-control register.

- CE0 Value of external pin CE0 (after it passes through an internal synchronizer). The value is not latched.
- **CE1** Value of external pin $\overline{CE1}$ (after it passes through an internal synchronizer). The value is not latched.
- **DE** Value of external pin DE (after it passes through an internal synchronizer). The value is not latched.
- AE Value of external pin AE (after it passes through an internal synchronizer). The value is not latched.

STRB0 SWW	Software wait states for STRB0 access. In conjunction with STRB0 WTCNT, this field defines the mode of wait-state generation. Actual wait states are explained in Section 9.4 and in Table 9–6.
STRB1 SWW	Software wait states for STRB1 access. In conjunction with STRB1 WTCNT, this field defines the mode of wait-state generation. Actual wait states are explained in Section 9.4 and in Table 9–6.
STRB0 WTCNT	Software wait-state count for $\overline{\text{STRB0}}$ accesses. Specifies the number of cycles to use when software wait states are active. Three-bit range is from 000 ₂ (zero) to 111 ₂ (seven).
STRB1 WTCNT	Software wait-state count for $\overline{\text{STRB}}$ 1 accesses. Specifies the number of cycles to use when software wait states are active. Three-bit range is from 000 ₂ (zero) to 111 ₂ (seven).
STRB0 PAGESIZE	Page size for STRB0 accesses. Specifies the number of MSBs of the address to use to define the bank size for STRB0 accesses. See ranges in Table 9–3 and subsection 9.3.2.
STRB1 PAGESIZE	Page size for STRB1 accesses. Specifies the number of MSBs of the address to use to define the bank size for STRB1 accesses. See ranges in Table 9–3 and subsection 9.3.2.
STRB ACTIVE	Specifies address ranges over which $\overline{\text{STRB0}^{\dagger}}$ and $\overline{\text{STRB1}^{\dagger}}$ are active. See ranges in Table 9–4 on for STRB ACTIVE and Table 9–5 for LSTRB ACTIVE.
STRB SWITCH	Inserts a single cycle between back-to-back reads that switch from $\overline{\text{STRB0}}$ to $\overline{\text{STRB1}}$ (or vice versa). When a 1, insert cycle. When a 0, <i>don't</i> insert cycle.
Reserved	Read as zeros.

STRBx PAGESIZE	External Address	External Address Bus Bits Defining	
(Bits 14–18, 19–23)‡	Bus Bits Defining the Current Page	Address on a Page	Page Size (32-Bit Wds)
00000-00110	Reserved	Reserved	Reserved
00111¶	30-8	7-0	2 ⁸ =256
01000	30-9	8-0	2 ⁹ =512
01001	30-10	9-0	2 ¹⁰ =1K
01010	30-11	10-0	2 ¹¹ =2K
01011	30-12	11-0	2 ¹² =4K
01100	30-13	12-0	2 ¹³ =8K
01101	30-14	13–0	2 ¹⁴ =16K
01110	30-15	14-0	2 ¹⁵ =32K
01111	30-16	15-0	2 ¹⁶ =64K
10000	30-17	16-0	2 ¹⁷ =128K
10001	30-18	17–0	2 ¹⁸ =256K
10010	30-19	18-0	2 ¹⁹ =512K
10011	30-20	19–0	2 ²⁰ =1M
10100	30-21	20-0	2 ²¹ =2M
10101	30-22	21-0	2 ²² =4M
10110§	30-23	22-0	2 ²³ =8M
10111	30-24	23-0	2 ²⁴ =16M
11000	30-25	24-0	2 ²⁵ =32M
11001	30-26	25-0	2 ²⁶ =64M
11010	30-27	26-0	2 ²⁷ =128M
11011	30-28	27-0	2 ²⁸ =256M
11100	30-29	28-0	2 ²⁹ =512M
11101	30	29-0	2 ³⁰ =1G
11110	None	30-0	2 ³¹ =2G
11111	Reserved	Reserved	Reserved

Table 9–3. Page Size as Defined by STRB0/1 PAGESIZE Bits[†]

[†] Mnemonics used are for the global-memory interface-control register. For the local-memory interface-control register, add the prefix L to the beginning of each mnemonic (e.g., LSTRB0 PA-GESIZE, LSTRB1 PAGESIZE, etc.). The description is the same for the local-memory interface-control register.

[‡]The x in STRBx means that the data in the columns are for STRB0 or STRB1.

§ A STRBx PAGESIZE field of 101102 is depicted in Figure 9–5 on page 9-13.

¶ Value at reset.

STRBx AC- TIVE Field (Bits 24–28)	STRB0 ACTIVE Address Range	Size of STRB0 ACTIVE Address Range	STRB1 ACTIVE Address Range
00000–01110	Reserved	Reserved	Reserved
01111	8000 0000-8000 FFFF	2 ¹⁶ =64K	8001 0000-FFFF FFFF
10000	8000 0000-8001 FFFF	2 ¹⁷ =128K	8002 0000-FFFF FFFF
10001	8000 0000-8003 FFFF	2 ¹⁸ =256K	8004 0000-FFFF FFFF
10010	8000 0000-8007 FFFF	2 ¹⁹ =512K	8008 0000-FFFF FFFF
10011	8000 0000-800F FFFF	2 ²⁰ =1M	8010 0000-FFFF FFFF
10100	8000 0000-801F FFFF	2 ²¹ =2M	8020 0000-FFFF FFFF
10101	8000 0000-803F FFFF	2 ²² =4M	8040 0000-FFFF FFFF
10110	8000 0000-807F FFFF	2 ²³ =8M	8080 0000-FFFF FFFF
10111	8000 0000-80FF FFFF	2 ²⁴ =16M	8100 0000-FFFF FFFF
11000	8000 0000-81FF FFFF	2 ²⁵ =32M	8200 0000-FFFF FFFF
11001	8000 0000-83FF FFFF	2 ²⁶ =64M	8400 0000-FFFF FFFF
11010	8000 0000-87FF FFFF	2 ²⁷ =128M	8800 0000-FFFF FFFF
11011	8000 0000-8FFF FFFF	2 ²⁸ =256M	9000 0000-FFFF FFFF
11100	8000 0000-9FFF FFFF	2 ²⁹ =512M	A000 0000-FFFF FFFF
11101	8000 0000-BFFF FFFF	2 ³⁰ =1G	C000 0000-FFFF FFFF
11110‡	8000 0000-FFFF FFFF	2 ³¹ =2G	None
11111	Reserved	Reserved	Reserved

[†] Address ranges specified by the LSTRB ACTIVE bits are listed in Table 9–5. [‡] Value at reset.

LSTRBx ACTIVE Field (Bits 24–28)	LSTRB0 ACTIVE Address Range	Size of LSTRB0 ACTIVE Address Range	LSTRB1 ACTIVE Address Range
00000–01110	Reserved	Reserved	Reserved
01111	0000 0000-0000 FFFF	2 ¹⁶ =64K	0001 0000-7FFF FFFF
10000	0000 0000-0001 FFFF	2 ¹⁷ =128K	0002 0000-7FFF FFFF
10001	0000 0000-0003 FFFF	2 ¹⁸ =256K	0004 0000-7FFF FFFF
10010	0000 0000-0007 FFFF	2 ¹⁹ =512K	0008 0000-7FFF FFFF
10011	0000 0000-000F FFFF	2 ²⁰ =1M	0010 0000-7FFF FFFF
10100	0000 0000-001F FFFF	2 ²¹ =2M	0020 0000 -7FFF FFFF
10101	0000 0000-003F FFFF	2 ²² =4M	0040 0000 –7FFF FFFF
10110	0000 0000-007F FFFF	2 ²³ =8M	0080 0000-7FFF FFFF
10111	0000 0000-00FF FFFF	2 ²⁴ =16M	0100 0000 –7FFF FFFF
11000	0000 0000-01FF FFFF	2 ²⁵ =32M	0200 0000 –7FFF FFFF
11001	0000 0000-03FF FFFF	2 ²⁶ =64M	0400 0000 – 7FFF FFFF
11010	0000 0000-07FF FFFF	2 ²⁷ =128M	0800 0000 –7FFF FFFF
11011	0000 0000 - 0FFF FFFF	2 ²⁸ =256M	1000 0000 –7FFF FFFF
11100	0000 0000-1FFF FFFF	2 ²⁹ =512M	2000 0000 –7FFF FFFF
11101	0000 0000 - 3FFF FFFF	2 ³⁰ =1G	4000 0000 –7FFF FFFF
11110‡	0000 0000 - 7FFF FFFF	2 ³¹ =2G	None
11111	Reserved	Reserved	Reserved

Address ranges below 0030 0000h are valid only in microprocessor mode (ROMEN=0). Access to reserved, peripheral, and on-chip memory areas does not activate LSTRB signals.
Value at reset.

9.3.1 Mapping Addresses to Strobes

Figure 9–4 demonstrates the relationship between the STRB ACTIVE bits (see Figure 9–3 on page 9-7 for more information) and the address ranges over which the signals, <u>STRB0</u> and <u>STRB1</u>, are active. Note that the address ranges of <u>STRBx</u> and <u>LSTRBx</u> also govern the ranges of associated signals—<u>RDYx</u>, <u>LRDYx</u>, R/Wx, LR/Wx, PAGEx, LPAGEx, etc. (where x=1 or 0).

Figure 9–4. Effects of STRB ACTIVE on Global Memory Bus Memory Map



NOTE: Shown here are two examples for the global memory map. The entire 'C40 memory map (local and global) is shown in Figure 4–1 on page 4-3. Note that the highest address for LSTRB1 (local bus) is 7FFF FFFFh.

Example (a) of Figure 9–4 shows the reset condition (STRB ACTIVE= 11110_2). In this case, signal STRB0 is active over the entire address range of the global memory bus (see Table 9–4 for fields and address ranges of STRB ACTIVE).

Example (b) of Figure 9–4 shows the global memory bus memory map when STRB ACTIVE=10101₂. In this case, STRB0 is active from addresses 8000 0000h-803F FFFFh, and STRB1 is active from addresses 8040 0000h-FFFF FFFFh (as shown in Table 9–4 for a STRB ACTIVE of 10101₂).

9.3.2 Page Size Operation

Within the memory range selected by any of the four strobe lines, the 'C4x external interface allows you to further divide the range into pages of selected length. This capability gives you great flexibility in the design of high-speed, high-density memory systems combined with slower peripheral devices; each time a page boundary is crossed, a cycle is inserted to allow external logic to reconfigure itself.

Each PAGESIZE field in the memory interface control register (shown in Figure 9–2 on page 9-7) works in the same manner to specify the page size for its corresponding strobe. Table 9–3 on page 9-9 illustrates the relationship between the PAGESIZE field and the bits of the address used to define the current page and the resulting page size. Page size begins at 256 words (with external address-bus bits 7–0 defining the address on a page, and ranges of up to 2G words ('C40) with external address bus bits 30–0 ('C40) defining the location on a page. The example in Figure 9–5 shows how a pagesize field value of 10110_2 is translated into bits 30-23 defining the current page and bits 22-0 defining an address on a page.

Figure 9–5. STRBx PAGESIZE Fields Example



Note: This figure represents a STRBx PAGESIZE field value of 101102 (as shown in Table 9-3).

Changing from one page to another causes a cycle to be inserted in the external access sequence, allowing external logic to reconfigure itself appropriately. For example, the extra cycle allows time for slower devices to get off the bus, thereby eliminating bus contention. The memory interface control logic keeps track of the address used for the last access for each STRB. When an access begins, the PAGE signal corresponding to the active STRB goes inactive (high) if the access is to a new page. The PAGE0 and PAGE1 signals are independent of one another, each having its own page-size logic.

At reset, the page-control logic is initialized so that the extra cycle is inserted for the first access to the two strobe interfaces.

The control registers for the local memory interface function in the same way as the control registers for the global memory interface.

9.4 Programmable Wait States

The 'C4x has its own internal software-configurable ready-generation capability for each strobe. This software wait-state generator is controlled by configuring two fields in the global or local interface control register. Use the STRBx WTCNT field (bits 8–10 and 11–13) to specify the number of software wait states to generate, and use the STRBx SWW field (bits 6–7, and 4–5) to select one of the following four modes of wait-state generation:

- External RDY (SWW = 0). Wait states are generated solely by the external RDY line (software wait-states ignored).
- WTCNT-generated RDY_{wtcnt} (SWW = 01₂). Wait states are generated solely by the software wait-state generator (external RDY ignored).
- Logical-OR of RDY and RDY_{wtcnt} (SWW = 10₂). Wait states are generated with a logical OR of internal and external ready signals. Either signal can generate ready.
- Logical-AND of RDY and RDY_{wtcnt} (SWW = 11₂). Wait states are generated with a logical AND of internal and external ready signals. Both signals must occur.

The four modes are used to generate the internal ready signal, \overline{RDY}_{int} , that controls accesses. As long as $\overline{RDY}_{int} = 1$, the current external access is extended. When $\overline{RDY}_{int} = 0$, the current access completes. Since the use of programmable wait states for both external interfaces is identical, only the global-bus interface is described in this section.

 $\overline{\text{RDY}}_{\text{wtcnt}}$ is an internally-generated ready signal. When an external access is begun, the value in WTCNT is loaded into a counter. WTCNT can be any value from 0 through 7. The counter is decremented every H1/H3 clock cycle until it becomes 0. Once the counter is cleared to 0, it remains cleared to 0 until the next access. When the counter is nonzero, $\overline{\text{RDY}}_{\text{wtcnt}}=1$. When the counter is 0, $\overline{\text{RDY}}_{\text{wtcnt}}=0$.

Table 9–6 is the truth table for each value of SWW, showing the different values at $\overline{\text{RDY}}$, $\overline{\text{RDY}}_{\text{wtcnt}}$, and $\overline{\text{RDY}}_{\text{int}}$.

Note:

At reset, the 'C4x inserts seven wait states for each access to external memory. These wait states are inserted to ensure that the system can function with slow memories. To increase system performance when using fast external memories, you will need to decrease the number of wait states.

SWW Value	RDY	RDY wtcnt	RDY int	RDY _{int}
00	0	0	0	RDY int is dependent only upon RDY.
00	0	1	0	RDY _{wtcnt} is ignored.
00	1	0	1	
00	1	1	1	
01	0	0	0	RDY int is dependent only upon
01	0	1	1	RDY _{wtcnt} . RDY is ignored.
01	1	0	0	With
01	1	1	1	
10	0	0	0	RDY int is the logical-OR (electrical
10	0	1	0	AND because these signals are low
10	1	0	0	true) of RDY and RDY wtcnt.
10	1	1	1	
11	0	0	0	RDY int is the logical-AND (electrical
11	0	1	1	OR because these signals are low
11	1	0	1	true) of RDY and RDY _{wtont} .
11	1	1	1	, within

9.5 Memory Interface Timing

Except for some cases that are covered in detail later in this chapter, the 'C4x offers a one-cycle external read and a pipeline external write. A write is considered a two-step operation: one cycle writes the data into the external memory port buffer and then another cycle moves the data from there to external memory.

Note:

From the perspective of the DMA or CPU, the write operation finishes in one cycle, and the DMA or CPU can proceed. However, if the next DMA or CPU access is to the same external bus, the DMA or CPU must wait, and the write is considered a two-cycle operation.

Figure 9–6. STRB and RDY Timing



Note: The dotted lines emphasize the relationships between the signals.

As shown in Figure 9–6, STRB changes on the falling edge of H1, and RDY is sampled on the falling edge of H1. Throughout the other timing diagrams in this section, the following general rules apply to the logical timing of the parallel external interfaces:

- \Box Changes of R/W are always framed by STRB.
- ☐ A page boundary crossing for a particular STRB results in the corresponding PAGE signal going high for one cycle.
- \square R/W transitions always occur on the rising edge of H1.
- STRB transitions always occur on the falling edge H1.
- **RDY** is always sampled on the falling edge H1.
- Data is always sampled during a read on the falling edge of H1.
- Data is always driven out during a write on the falling edge of H1.

- Data is always stopped from being driven during a write on the rising edge of H1.
- The status and PAGE signals, following a read, change on the falling edge of H1. The address also changes on H1's falling edge.
- The status and PAGE signals, following a write, change on the falling edge of H1; the address changes on the rising edge of H1.
- The fetch of an interrupt vector over an external interface is identified by the status signals for that interface (STAT or LSTAT) as a data read.
- The interlocked operation status signals (LOCK and LLOCK) have the same timing as the STAT and LSTAT status signals, respectively.
- Any time PAGE goes high, STRB goes high.

Note:

When no external port is accessing memory (idle status), the control lines are inactive (\overline{RDY} is ignored, STRB is high, and the STATx lines become high), the address lines keep the last value used in the pins, and the data lines become high-impedance. This can be seen in Figure 9–16.

Figure 9–7 illustrates a read, read, write sequence. This figure assumes that all three accesses are to the same page and that they are STRB1 accesses. This timing diagram illustrates that:

- Back-to-back reads to the same page are single-cycle accesses.
- STRB stays low during back-to-back reads.
- ☐ When the transition from a read to a write is done, STRB goes high for one cycle to frame the R/W signal changing.



Figure 9–7. Read Same Page, Read Same Page, Write Same Page Sequence

Note: Strobe and Ready Further Defined

Strobe and ready are discussed from the application viewpoint in TMS320C4x General-Purpose Applications User's Guide.

Figure 9–8 shows that:

- ☐ To prevent unwanted writes, STRB goes high between back-to-back writes to disable the memory while the address changes.
- ☐ As in Figure 9–7, STRB goes high between a write and a read, and it frames the R/W transition.
- A read following a write on the same bus takes two cycles. This happens regardless of whether or not the read is on the same strobe and/or page.
- Consecutive writes take two cycles.

Figure 9–8. Write Same Page, Write Same Page, Read Same Page Sequence



Figure 9–9 shows that going from one page to another on back-to-back reads causes:

- An extra cycle to be inserted to allow the next memory to be selected
- The transition to be signaled by PAGE going high for one cycle
- □ STRB1 to go high for one cycle

Figure 9–9. Read Same Page, Read Different Page, Read Same Page Sequence



Figure 9–10 shows that on back-to-back writes, when a page switch occurs:

- □ PAGE1 signals this occurrence by going high for one cycle.
- □ No extra cycle is inserted, because write cycles exhibit an inherent onehalf H1 cycle setup of address information before STRB goes low.

Figure 9–10. Write Same Page, Write Different Page, Write Same Page Sequence





Figure 9–11. Write Same Page, Read Different Page, Write Different Page Sequence



Figure 9–12. Read Different Page, Read Different Page, Write Same Page Sequence

External Bus Operation 9-23



Figure 9–13. Write Different Page, Write Different Page, Read Same Page Sequence



Figure 9–14. Read Same Page, Write Different Page, Read Different Page Sequence

External Bus Operation 9-25
Figure 9–15 through Figure 9–19 illustrate idle bus cycles. Idle bus cycle timing is similar to read cycle timing. The primary differences are that no data is read, $\overline{\text{STRB}}$ is held high, and $\overline{\text{RDY}}$ is ignored.

Figure 9–15. Read Same Page, Idle One Cycle, Read Same Page Sequence



9-26



Figure 9–16. Write Same Page, Idle One Cycle, Write Different Page Sequence

External Bus Operation 9-27

Figure 9–17. Idle, Read Different Page, Idle Sequence





Figure 9–18. Idle, Write Same Page, Idle Sequence

External Bus Operation 9-29



Figure 9–19. Write Different or Same Page, Idle, Idle Sequence

Figure 9–20 illustrates a STRB1 read followed by a STRB0 read when STRB SWITCH=0. This mode allows the reads to be back-to-back, with no cycles inserted between them when they are activating different strobes.

Figure 9–20. Read Same Page on $\overline{STRB1}$, $\overline{STRB0}$, and on $\overline{STRB1}$ Sequence When \overline{STRB} SWITCH = 0



Figure 9–21 is similar to Figure 9–20 except that the second STRB1 read is from a different page than the first.

Figure 9–21. Read Same Page on STRB1, STRB0, Read Different Page on STRB1 Sequence When STRB SWITCH = 0



Figure 9–22 illustrates a STRB1 read followed by a STRB0 read when STRB SWITCH=1. In this mode, a cycle is inserted between back-to-back reads that activate different strobes. Some memory configurations require this cycle between strobe transitions to prevent bus conflicts during back-to-back reads on different strobes.

Figure 9–22. Read Same Page on STRB1, STRB0, and on STRB1 Sequence When STRB SWITCH = 1



External Bus Operation 9-33

Figure 9–23 is similar to Figure 9–22 except that the second STRB1 read is from a different page than the first.

Figure 9–23. Read Same Page on STRB1, STRB0, Read Different Page on STRB1 Sequence When STRB SWITCH = 1







External Bus Operation 9-35

Figure 9–25 and Figure 9–26 show one wait-state read and write operations, respectively.

Figure 9–25. Read With One Wait State



9-36





9.6 Using Enable Signals to Control Signal Groups

Figure 9–27. Using Enable Signals to Put Signal Groups in a High-Impedance State



Figure 9–27 shows an enable signal controlling the corresponding signal group. For example, signal $\overline{\text{DE}}$ controls the global external-interface data signals. The enable signals are unsynchronized inputs that turn off the corresponding output buffers. After the enable signal goes high plus timing (1) in Figure 9–27, the corresponding signal group goes into high-impedance. Then, after the enable signal goes low plus timing (2) in Figure 9–27, the signal group comes out of high-impedance. If the signal group is already in a high-impedance state before the enable signal goes high, the group will come out of the high-impedance state (when the enable signal goes low) only if the signal group is in a state requiring it to do so. For example, a data bus that was not being driven will be driven after being enabled, if an access is pending for the data bus.

Note:

If you intend to use internally generated wait states, be certain that no data is read from or written to the bus when it is disabled. This is because it is possible for a bus to be in the high-impedance state with internally generated wait states. In this case, data that is written will not be seen externally, and data that is read will be whatever value is sampled on the high-impedance bus.

9.7 Interlocked Operations

One of the most common parallel processing configurations is the sharing of global memory by multiple processors. For multiple processors to access this global memory and share data in a coherent manner, some sort of arbitration or handshaking is necessary. 'C4x interlocked operations meet this requirement for arbitration. More details are given in Section 9.7.5 on page 9-44.

Five 'C4x instructions are referred to as interlocked operations. Through the use of external signals, these instructions provide powerful synchronization mechanisms. They also guarantee integrity of communication and result in a high-speed operation. The interlocked-operation instruction group is listed in Table 9–7.

Table 9–7. Interlocked Operations

Instruction	Description	Operation
LDFI	Load floating-point value from memory into a register; interlocked when <i>external</i> memory accessed	Signal interlocked src \rightarrow dst
LDII	Load integer from memory into a register; in- terlocked when <i>external</i> memory accessed	Signal interlocked src \rightarrow dst
SIGI	Load floating-point value from memory into a register; interlocked when <i>external</i> memory accessed	Signal interlocked Clear interlock
STFI	Store floating-point value from a register to memory; interlocked when <i>external</i> memory accessed	$src \rightarrow dst$ Clear interlock
STII	Store integer from a register to memory; inter- locked when <i>external</i> memory accessed	$src \rightarrow dst$ Clear interlock

The interlocked operations use the global- and local-bus pins, \overline{LOCK} and \overline{LLOCK} , to reflect a currently executing interlocked operation. This signal is active (low) when any of the interlocked instructions in Table 9–7 are executing.

The external timing for interlocked loads and stores is the same as for standard loads and stores. You can extend interlocked loads and stores like standard accesses by using the appropriate ready signal (RDYx or LRDYx).

9.7.1 LDFI and LDII

The LDFI and LDII instructions perform the following actions:

- 1) Pull (L)LOCK low.
- 2) Execute an LDF or LDI instruction.
- Extend the read cycle until the appropriate ready signal is received. Complete the instruction.
- 4) Leave (L)LOCK active low until changed by an STFI, STII, or SIGI.

The read/write operation is identical to any other read/write cycle except for the special use of $\overline{(L)LOCK}$. The *src* operand for LDFI and LDII is always a direct or indirect memory address. $\overline{(L)LOCK}$ is set to 0 only if the *src* is located off-chip (i.e., STRB or LSTRB is active). If on-chip memory is accessed, then $\overline{(L)LOCK}$ is not asserted, and the operation is as an LDF or LDI from internal memory.

9.7.2 STFI and STII

The STFI and STII instructions perform the following operations:

- Begin a write cycle. The state of (L)LOCK does not change. If it is low, an interlocked operation occurs. If high, the operation is as if an STF or STI is performed (not interlocked).
- Execute an STF or STI instruction and extend the write cycle until the appropriate ready is signaled.
- 3) After the write cycle, bring $\overline{(L)LOCK}$ inactive (high).

As in the case for LDFI and LDII, the *dst* of STFI and STII affects (L)LOCK. If *dst* is located off-chip (STRB(0,1) or LSTRB(0,1) is active), (L)LOCK is set to a 1. If on-chip memory is accessed, then (L)LOCK is not asserted, and the operations are as a STF or STI to internal memory.

9.7.3 SIGI

The SIGI instruction can be used in a variety of ways. In some applications, you may wish to modify semaphores externally, perhaps with special-purpose logic. If so, SIGI can be used to perform a single-cycle interlocked access of the semaphore. The SIGI instruction can also be used simply to perform an external read and to signal that a particular point in your code has been reached.

The SIGI instruction functions as follows:

- 1) Pulls (L)LOCK low
- 2) Executes an LDI instruction
- Extends the read cycle until the appropriate ready signal is received. Completes the instruction
- 4) Brings (L)LOCK back inactive high

Interlocked operations can be used to implement a busy-waiting loop, to manipulate a multiprocessor counter, to implement a simple semaphore mechanism, or to perform synchronization between two 'C4xs. The following examples illustrate the usefulness of the interlocked operations instructions.

9.7.4 Interlocked Examples

Examples in this section show you how interlocked operations can be used to implement:

- ☐ A busy-waiting loop to synchronize processors at the software level (Example 9–1, page 9-42)
- A counter shared between cooperative processors that defines the number of times a task should be done by the processors (Example 9–2 on page 9-42)
- Semaphores to ease the programming of critical sections (Example 9–3 and Example 9–4 on page 9-43)

Example 9–1 shows the implementation of a busy-waiting loop. The 'C4x stays in this loop until another processor writes a 0 in @LOCK. If location LOCK is the interlock for a critical section of code, and a nonzero means the lock is busy, the algorithm for a busy-waiting loop can be used as shown.

Example 9–1. Busy-Waiting Loop

	LDI	1,R0	;Put 1 in RO
L1:	LDII	@LOCK,R1	;Load lock value into R1
	STII	R0,@LOCK	;Set lock value to 1
	BNZ	L1	; If R1 (previous lock value) is not
			;0, read it again

Example 9–2 shows how a location COUNT may contain a count of the number of times a particular operation must be performed. This operation may be performed by any processor in the system. If the count is zero, the processor waits until it is nonzero before beginning processing. The example also shows the algorithm for modifying COUNT correctly.

Example 9–2. Task Counter Manipulation

	LDI	0,R0	
WAIT	LDII	@COUNT,R1	;Read current value of counter
	BZD	WAIT	; If COUNT = 0, try again
	LDNZ	1,R0	; If COUNT not zero, decrement it
	SUBI	R0,R1	
	STII	R1,@COUNT	;Update COUNT

Figure 9–28 illustrates multiple 'C4xs sharing global memory and using interlocked instructions as shown in Example 9–3 and Example 9–4.

Figure 9–28. Multiple 'C4x Devices Sharing Global Memory



Example 9–3. Implementation of V(S)

```
V: LDII @S,R0
ADDI 1,R0
STII R0,@S ; S + 1 → S
```

Example 9–4. Implementation of P(S)

```
LDI
         0,R0
P: LDII
         @S,R1
                   ;Read semaphore's current value
                   ; If S = 0, go to P and try again
   BZD
         Ρ
                   ; If S is not 0, decrement it
   LDNZ
         1,R0
   SUBI
         R0,R1
   STII
         R1,0S
                   ;Update S
```

Sometimes it may be necessary for several processors to access some shared data or other common resources. The portion of code that must access the shared data is called a *critical section*.

To ease the programming of critical sections, semaphores may be used. Semaphores are variables that can take only nonnegative integer values. Two primitive, indivisible operations are defined on semaphores (with S being a semaphore):

Indivisibility of V(S) and P(S) means that when these processes access and modify the semaphore S, they are the only processes doing so.

To enter a critical section, a P operation is performed on a common semaphore, for example, on S (S is initialized to 1). The first processor performing P(S) will be able to enter its critical section. All other processors are blocked because S has become 0. After leaving its critical section, the processor performs a V(S), thus allowing another processor to execute P(S) successfully.

The 'C4x code for V(S) is shown in Example 9–3, and code for P(S) is shown in Example 9–4. Compare the code in Example 9–4 to the code in Example 9–2, which does not use semaphores.

9.7.5 Bus-Lock Pins and Bus Timing

The timing of the $\overline{\text{LOCK}}$ and $\overline{\text{LLOCK}}$ pins is the same as the timing of the STAT(3–0) and LSTAT(3–0) pins. The LDII, LDFI, ,STII, STFI, and SIGI instructions manipulate the bus-lock signals *only* when an external memory access is made.

LDII, LDFI, and SIGI all clear \overline{LOCK} or \overline{LLOCK} to zero at the beginning of the read cycle with H1 falling. STII, STFI, and SIGI all set \overline{LOCK} or \overline{LLOCK} to one at the end of the access cycle on the falling edge of H1. Interlocked instructions are explained in Section 9.7.

Figure 9–29 through Figure 9–32 show bus timing characteristics for several external accesses using STII, LDII, STFI, LDFI, and SIGI.





Figure 9–29 is an example of an LDII or LDFI external access.

External Bus Operation 9-45

Figure 9–30 is an example of STII or STFI external access following the previous interlocked load (shown in Figure 9–29) and an idle cycle. This is the timing for an interlocked load/interlocked store sequence.

Figure 9–30. LDII or LDFI and STII or STFI External Access





Figure 9–31 is an example of a SIGI external access.

Figure 9–31. SIGI External Access Timing

Figure 9–32 illustrates timing for SIGI if the $\overline{\text{LOCK}}$ signal is already low. This could occur when a SIGI follows an LDII instruction. Since $\overline{\text{LOCK}}$ is already low, the only effect SIGI has on $\overline{\text{LOCK}}$ is to bring it high.

Figure 9–32. SIGI When LOCK Is Already Low



9.8 IACK Timing

The IACK pin is affected by the IACK (interrupt acknowledge) instruction. The timing of the pin is similar to that of the \overline{LOCK} pin when used by the SIGI instruction. In all respects (timing, extension with wait states, etc.) the IACK behaves like a \overline{LOCK} or STAT signal. The only difference is that there is only one IACK pin.

The timing for the IACK pin is shown in Figure 9–33. Like the interlocked instructions, the IACK instruction affects IACK *only* for an external access.



Figure 9–33. IACK Timing

Chapter 10

Page

The Bootloader

The bootloader provided in the on-chip ROM of the 'C4x can load and execute source programs that are received from a host processor, an EPROM, or a standard memory device. The 'C4x bootloader functions primarily as either a *memory bootloader* or as a *communication port bootloader*.

Topic

10.1Bootloader Description10-210.2Mode Selection10-310.3Bootloading Sequence10-510.4Bootloading from External Memory (Examples)10-1010.5Bootloading from a Communication Port (Examples)10-1610.6Modifying the IIOFx Pins After Bootloading10-1910.7The Bootloader Program10-20

10.1 Bootloader Description

The bootloader code starts at location 0x11bc in the on-chip ROM in both the 'C40 and 'C44. For 'C44 device revisions ≤ 1.0 , the 'C44 bootloader code is identical to the 'C40 bootloader code. For 'C44 device revisions > 1.0, the 'C44 bootloader code differs in three memory locations from the 'C40 bootloader. These three locations are noted in the code. The bootloader program is listed in Section 10.7, *The Bootloader Program*.

10.2 Mode Selection

The 'C4x bootloader functions primarily as either a *memory bootloader* or a *communication port bootloader*. Bootloader mode selection is determined by the IIOF(3–0) pins, as described in Table 10–1 and shown in Figure 10–1.

- □ The memory bootloader supports user-definable byte, half-word, and full-word data formats, which allow the flexibility to load a source program from memories having widths of 8 bits, 16 bits, or 32 bits. The source programs to be loaded must reside in one of six predefined memory locations, as listed in Table 10–1. STRB0 (LSTRB0) should be used because they are the active strobes after reset. Figure 10–2 shows the flow for the memory bootloader.
- □ The communication port bootloader waits for the first data input from one of the six ('C40) or four ('C44) communication port channels and uses that channel to perform the bootload. The format of the incoming data stream is similar to that for a memory data stream, except that the source memory width is excluded (the format is described in Table 10–2). Figure 10–3 shows the flow of the communication port bootloader.

External Pin				Source Program Location	
IIOF3	IIOF2	IIOF1	liOF0	'C40	'C44
1	1	0	1	0030 0000h	0030 0000h
1	0	1	1	4000 0000h	4000 0000h†
1	0	0	1	6000 0000h	0080 0000h
0	1	1	1	8000 0000h	8000 0000h†
0	1	0	1	A000 0000h	8040 0000h†
0	0	1	1	C000 0000h	8080 0000h†
0	0	0	1	Reserved (the boot- loader terminates)	Reserved (the boot- loader terminates)
1	1	1	1	Communication port	Communication port

Table 10–1. Bootloader Mode Selection Using Pins IIOF(3–0)

[†] The 'C44 external-address buses each have only the low 24 bits of the internal address bus. Thus, the internal address 4000000h maps to 0h on the local bus. Any address at or above 80000000h is mapped to the global bus; 80800000, for example, maps to address 00800000h on the global bus.



Figure 10–1. Mode Selection Flow

10.3 Bootloading Sequence

Here is the general sequence of events in bootloading a source program:

- Select the bootloader by resetting the 'C4x while driving the RESETLOC(1,0) pins low, the on-chip ROM enable pin (ROMEN) high, and the IIOF0 pin high. The ROMEN pin must be high during bootloader execution, but it can be changed anytime after bootloading.
- 2) The status of external pins IIOF(3–1) indicates where to find the source program to be loaded (memory or communication port). These options are listed in Table 10–1. Pins IIOF(3–1) are read as the IIOF flags in the CPU IIF register. The bootloader takes the following steps to determine the source program's location, as is shown in Figure 10–1.
 - a) If an IIF(3–1) value of from 110₂ to 001₂ (6 to 1) is found, the source program is loaded from the corresponding memory address shown in the top six lines of Table 10–1. See Figure 10–2 for details on boot-loader memory flow.
 - b) The IIF(3–1) value of 000₂ (0) is reserved. You should not use this mode.
 - c) If none of the combinations 000₂ 110₂ are found, the bootloader program assumes that loading will be via a communication port, and it starts checking communication port input channels (in the order port 0 through port 5). If it finds no inputs from a communication port, the program returns to checking the status of the IIOF(3–1) pins again. See Figure 10–3 for details on bootloader communication port flow.
- 3) When the source program's data stream is found, the program is loaded at the address found in the fifth word of the data stream (the format is shown in Table 10–2), using the bus width specified in the first word (8, 16, or 32 bits wide). The bootloader cannot load the source program to any location below 0000 1000h, unless the address decode logic is remapped. The first five words of the source program specify its loading and execution criteria. Remaining words are the source program(s) and vector table pointers as shown in Table 10–2.
- 4) An IACK instruction is executed, indicating the completion of the bootload sequence. This indication can then be used to switch from microcomputer (ROMEN = 1) to microprocessor mode (ROMEN = 0). You do not need to reset the 'C4x to change the ROMEN pin. However, ensure that the 'C4x will not access addresses 0000 0000h to 0000 0FFFh during the change.
- 5) The source program is executed (entry point is the first word of the *first* loaded program).







Figure 10–3. Communication-Port Load Mode Flow

The Bootloader 10-7

The data stream with its source program(s) should be in the format shown in Table 10–2. The contents of words 4 through *n* vary for the different source programs loaded throughout the entire data stream. The first three words and the last three words are nonvariables that affect each of the source-program blocks. The eight least significant bits (LSBs) of the first word specify the memory width. If byte or half-word wide is selected, the loading sequence is from LSBs to MSBs.

Table 10–2. Structure of Source Program Data Stream

Word	Contents
1	Memory width where source program resides (8, 16, or 32 bits wide)
2	Value to set in the global memory interface control register (shown in Figure 9–2).
3	Value to set in the local memory interface control register (shown in Figure 9–2).
4	Block size in 32-bit words of the first program block to be loaded (after the number of words is loaded, the next word should be all zeros; if not, another block is assumed to follow).
5	Address where the source program is to be loaded.
6	First word of source program.
n	Last word of source program (the program organized as words 4 through n — these shaded words).
<i>n</i> +1	Word of all zeros. (Note that if several source-program blocks were sent, word <i>n</i> above would be the last word of the <i>last</i> source-program block. Each source-program block would have the format shown in words 4 through <i>n</i> . This word of all zeros follows the <i>last</i> source program block).
<i>n</i> +2	IVTP value (interrupt vector table pointer, see Section 3.2).
<i>n</i> +3	TVTP value (trap vector table pointer, see Section 3.2).
<i>n</i> +4	Memory location for IACK instruction (see IACK instruction in Chapter 14).
Note: Th	ne shaded area identifies the source program block.

Each source program in a multiple block program transfer can be loaded to different specified destinations. Each program block specifies its program's size and destination address at the beginning of the block. End the entire block program loader function by following the last block with an all-zero word (0000 0000h). The second and third last words of the source memory define the interrupt vector table pointer (IVTP) and the trap vector table pointer (TVTP). The last word of the source memory defines the memory location for the IACK instruction. The IACK instruction brings the IACK signal low as data is read, if the memory location specified in the IACK instruction is in external memory that is available in the system. Finally, the processor begins execution of the first code block.

It is assumed that at least one block of source will be loaded when the bootloader is invoked. Initial loader invocation with a block size of 0000 0000h produces unpredictable results.

10.4 Bootloading from External Memory (Examples)

When the 'C4x's ROMEN input pin is high and RESETLOC(1,0)= 00_2 during reset, the memory bootloader can load programs stored in off-chip memory (typically 8-, 16-, or 32-bit ROMs) at an address determined by the \overline{IIOF} pins to any valid external or internal memory in the 'C4x's memory map.

Because address zero (0) is reserved for the bootloader, address zero should not be used for the reset vector when a user-defined, internal ROM-code mask is used.

The 8 LSBs of the first word of data read stream specify the memory width (8, 16, or 32 bits) as shown in Table 10–3, Table 10–4, and Table 10–5.

8-bit memories:	08h
16-bit memories:	0010h

32-bit memories: 0000 0020h

If 8- or 16-bit external memories are used, the loading sequence is from LSBs to MSBs. The bootloader reads the contents of 16-bit wide memories (least significant half word first) and packs each pair of 16-bit half words to make a 32-bit word before loading each word to memory. Accordingly, the bootloader reads the contents of byte-wide memories (least significant byte first) and packs each group of four bytes into a 32-bit word before loading each word to memory. Because the bootloader packs bytes before loading, no external hardware is needed to pack the loaded bytes into a 32-bit word. For 32-bit wide external memories, no byte packing is necessary, because the memory data width matches that of the 'C4x.

For 16-bit memories, the data read is expected to be in bit positions 0–15. Thus, the half-word memory's data lines should be interfaced to 'C4x data lines (L)D15–0. For byte-wide memories, the data read is expected to be in bit positions 0–7. Hence, the byte-wide memory's data lines should be interfaced to 'C4x data lines (L)D7–0. Even though the 'C4x does not require that unused data lines be pulled up to V_{CC}, it is recommended that each unused data line be pulled up through separate 22 K Ω resistors to 5 volts for minimum power dissipation.

Table 10–3, Table 10–4, and Table 10–5 show example data streams for 8-bit, 16-bit, and 32-bit wide configured memories, respectively.

These examples assume that:

- The status of the $\overline{IIOF}(0-3)$ pins is 110_2 after reset is deasserted (memory load from 0030 0000h see Table 10–1).
- The source program resides at memory location 0030 0000h and defines the following:
 - Memory width for bootloader: 8, 16, or 32 bits
 - Global bus memory with one software wait state, external RDY (SWW = 11), page size = 64K words for both STRB0 and STRB1, and an active address range = 1G words for both STRB0 and STRB1.
 - Local memory bus that requires two software wait states (SWW = 01), page size = 32K words, and active address range = 1G words for both STRB0 and STRB1.
 - First block program of 294 words in length and whose destination address is at 002F F840h.
 - Second block program of 64 words in length and whose destination address is at 002F F800h.
 - IVTP and TVTP, which are overlapped and point to the beginning of the on-chip RAM.
 - Memory location of 0030 0000h for IACK instruction.

Table 10–3. Byte-Wide Configured Memory

Word	Address	Value	Comments
1	0030 0000h	08h	Memory width = 8 bits
	0030 0001h	00h	
	0030 0002h	00h	
	0030 0003h	00h	
2	0030 0004h	F0h	Global memory bus control word = 1D7B C9F0h
	0030 0005h	C9h	(Described in Figure 9–2 on page 9-7)
	0030 0006h	7Bh	
	0030 0007h	1Dh	
Table 10–3. Byte-Wide Configured Memory (Continued)

Word	Address	Value	Comments
3	0030 0008h	50h	Local memory bus control word = 1D73 9250h
	0030 0009h	92h	(Described in Figure 9–2 on page 9-7)
	0030 000Ah	73h	
	0030 000Bh	1Dh	
4	0030 000Ch	26h	1st source program block size = 126h
	0030 000Dh	01h	
	0030 000Eh	00h	
5	0030 0010h	40h	1st source program block starting addr = 002F F840h
	0030 0011h	F8h	
	0030 0012h	2Fh	
	0030 0013h	00h	
6	0030 0014h		1st source program block starts here (first word)
to 299	•		•
	● 0030 04ABh		 1st source program block ends here (last word)
300	0030 04ACh	40h	2nd source program block size = 40
	0030 04ADh	00h	
	0030 04AEh	00h	
	0030 04AFh	00h	
301	0030 04B0h	00h	2nd source program block starting addr = 002F F800h
	0030 04B1h	F8h	
	0030 04B2h	2Fh	
	0030 04B3h	00h	
302	0030 04B4h		2nd source program block starts here (first word)
to	•		•
365	•		•
	0030 05B3h		2nd source program block ends here (last word)

Note: The shaded area identifies the source program block.

-

Word	Address	Value	Comments
366	0030 05B4h	00h	Value 0 to terminate the program block load
	0030 05B5h	00h	
	0030 05B6h	00h	
	0030 05B7h	00h	
367	0030 05B8h	00h	IVTP = 002F F800h
	0030 05B9h	F8h	
	0030 05BAh	2Fh	
	0030 05BBh	00h	
368	0030 05BCh	00h	TVTP = 002F F800h
	0030 05BDh	F8h	
	0030 05BEh	2Fh	
	0030 05BFh	00h	
369	0030 05C0h	00h	Memory location for IACK instruction =0030 0000h
	0030 05C1h	00h	
	0030 05C2h	30h	
	0030 05C3h	00h	(This is the final word in the data stream.)

Table 10–3. Byte-Wide Configured Memory (Continued)

Note: The shaded area identifies the source program block.

Word	Address	Value	Comments
1	0030 0000h	0010h	Memory width = 16 bits
	0030 0001h	0000h	
2	0030 0002h	C9F0h	Global memory bus control word = 1D7B C9F0h
	0030 0003h	1D7Bh	
3	0030 0004h	9250h	Local memory bus control word = 1D73 9250h
	0030 0005h	1D73h	
4	0030 0006h	0126h	1st program block size = 126h
	0030 0007h	0000h	
5	0030 0008h	F840h	1st program block starting addr.= 002F F840h
	0030 0009h	002Fh	
6 to 299	0030 000Ah ● ●		1st program block starts here (first word) • •
	• 0030 0255h		• 1st program block ends here (last word)
300	0030 0256h	0040h	2nd program block size = 40h
	0030 0257h	0000h	
301	0030 0258h	F800h	2nd program block starting addr.= 002F F800h
	0030 0259h	002Fh	
302 to 365	0030 025Ah • • 0030 02D9h		2nd program block starts here (first word) • • 2nd program block ends here (last word)
366	0030 02DAh	0000h	Value 0 to terminate the program block load
	0030 02DBh	0000h	
367	0030 02DCh	F800h	IVTP = 002F F800h
	0030 02DDh	002Fh	
368	0030 02DEh	F800h	TVTP = 002F F800h
	0030 02DFh	002Fh	

Table 10-4.16-Bit Wide Configured Memory

Note: The shaded area identifies the source program block.

Table 10-4.16-Bit Wide Configured Memory (Continued)

Word	Address	Value	Comments
369	0030 02E0h	0000h	Memory location for IACK instruction = 0030 0000h
	0030 02E1h	0030h	(This is the final word in the data stream.)

Note: The shaded areas identify the source program blocks.

Table 10–5.32-Bit Wide Configured Memory

Word	Address	Value	Comments
1	0030 0000h	0000 0020h	Memory width = 32 bits
2	0030 0001h	1D7B C9F0h	Global memory bus control word = 01D7B C9F0h
3	0030 0002h	1D73 9250h	Local memory bus control word = 01D73 9250h
4	0030 0003h	0000 0126h	1st program block size = 126h
5	0030 0004h	002F F840h	1st program block starting addr = 002F F840h
6 to 299	0030 0005h • • 0030 012Ah		1st program block starts here (first word) • • 1st program block ends here (last word)
300	0030 012Bh	0000 0040h	2nd program block size = 40h
301	0030 012Ch	002F F800h	2nd program block starting addr = 002F F800h
302 to 365	0030 012Dh • • 0030 016Ch		2nd program block starts here (first word) • • • 2nd program block ends here (last word)
366	0030 016Dh	0000 0000h	Value 0 to terminate the program block load
367	0030 016Eh	002F F800h	IVTP = 002F F800h
368	0030 016Fh	002F F800h	TVTP = 002F F800h
369	0030 0170h	0030 0000h	Address location for IACK instruction = 00030 0000h

Note: The shaded areas identify the source program blocks.

10.5 Bootloading from a Communication Port (Examples)

A value of all 1s on $\overline{IIOF}(0-3)$ signals that the source program is being transmitted via a communication port. Bringing all four of the $\overline{IIOF}(0-3)$ pins high also allows the pins to be used as interrupt lines without any external decode logic. With pins $\overline{IIOF}(0-3)$ all high at reset, the 'C4x polls the input level of each port to determine which channel contains the program. The input data sequence of the communication bootloader is the same as that of the memory bootloader except that it lacks the source memory width definition (because the memory width of the communication port bootloader is fixed).

Example 10–1 is a program listing for booting a multiprocessor system.

After a 32-bit boot from external memory, the master 'C4x boots — via a communication port—another 'C4x (slave processor) connected to communication port 0 of the master processor. Both processors stay in an infinite loop after booting. The code should be loaded in the master 'C4x EPROM in the correct memory location according to the IIOF settings of the master 'C4x. All IIOF pins of the slave processor should be set to 1. The ROMEN pin is enabled (ROMEN=1) and the RESETLOC(1,0) pins are low in both processors. For a description of how to convert an executable COFF file into an EPROM programmer format, see the hex conversion utility in the *TMS320 Floating Point Assembly Language Tools User's Guide* (literature number SPRU035).

```
Example 10–1. Booting a 'C4x Multiprocessor System
```

*_____ MASTER PROCESSOR BOOT TABLE _____ .text ; memory width .word 32 .word 3003c000h ; MASTER global control register ; (system specific !!) .word 3d79c210h ; master local control register ; (system specific !!) *_____ _____ MASTER PROCESSOR PROGRAM BLOCK *_____ ; block size .word 10 .word 2ff800h ; block dest addr * Code for master processor: this code sends boot table to slave processor ldi 8,rc ; loop 9 times: size of slave processor ; boot table rptbd endbl ldp src ; src in external memory ldi @src,ar0 ldi @dst,ar1 ldi *ar0++(1),r0 ; block start endb1: sti r0,*ar1 bu \$; master processor loops forever .word BOOT_TABLE2 ; address of boot table of slave src ; processor dst .word 100042h ; address of OFIFO connected to slave ; processor *_____ END OF ALL BLOCKS *_____ ; master end of bootload sequence ; master IVTP value .word 0 .word 2ffd00h .word 2ffd00h ; master TVTP value ; master address for iack .word 4000000h END OF MASTER PROCESSOR BOOT TABLE : size = 9 words *_____

SOOT TABLE2:		: slave BOOT TABLE
.word	3003c000h	; slave global control register
		; (system specific !!!)
.word	3d79c210h	; slave local control register
		; (system specific !!!)
.word	1	; block size
.word	2ff800h	; dst load address
bu	\$; slave processor loops forever
.word	0	; slave end of bootload sequence
.word	2ffd00h	; slave IVTP value
.word	2ffd00h	; slave TVTP value
.word	40000000h	; slave address for iack

Example 10–1. Booting a 'C4x Multiprocessor System (Continued)

10.6 Modifying the IIOFx Pins After Bootloading

The load options are based upon the status of $\overline{IIOF}(3-0)$ as general-purpose input pins. Therefore, to select the correct bootloader mode, pins $\overline{IIOF}(3-0)$ must be kept at a constant valid status value (see Table 10–1 for a list of values).

After the bootload is complete, the \overline{IACK} signal is brought low until the read phase in the pipeline finishes. Figure 10–4 shows an example circuit that generates the $\overline{IIOF}(3-0)$ signals for bootload selection and, after bootload operation, allows incoming external interrupts. In this example, after reset, the \overline{IIOF} pins stay low until the \overline{IACK} signal is received.

Figure 10–4. Circuit for Generation of a Low IIOF Signal for Bootloader Selection



10.7 The Bootloader Program

* C40BOOT - TMS320C40 BOOTLOADER PROGRAM (C) COPYRIGHT TEXAS INSTRUMENTS INC., 1990 * NOTE 1. AFTER THE DEVICE IS RESET, THE PROGRAM IS CHECKING THE INPUT STATUS OF IIOF0-3 PINS AND COMMUNICATION PORT INPUT FLAGS TO CONFIGURE ITSELF WHEN THE ON-CHIP ROM IS ENABLED (ROMEN=1). THE IIOFO PIN IS ASSUMED TO BE HIGH. * NOTE 2. THE FUNCTION SELECTION OF IIOF0-3 IS LISTED AS: IIOF IIOF IIOF IIOF FUNCTION 3 2 1 0 Memory bootloader from 00300000H 1 1 0 1 1 0 1 1 Memory bootloader from 4000000H 1 0 0 1 Memory bootloader from 6000000H 0 1 1 1 Memory bootloader from 8000000H 0 1 0 1 Memory bootloader from A000000H 0 0 1 1 Memory bootloader from C000000H 0 0 0 1 Reserved 1 1 1 1 Communication port bootloader * THE PROGRAM ASSUMES THE COMMUNICATION PORT BOOTLOADER IS THE DEFAULT FUNCTION. IF NO OTHER FUNCTION IS SELECTED, THE PROGRAM STARTS CHECKING THE COMMUNICATION PORT INPUT CHANNELS. IF THERE IS NO INPUT FROM A * COMMUNICATION PORT, THE PROGRAM RECHECKS THE IIOF (3-0) STATUS AGAIN. * NOTE 3.MEMORY BOOTLOADER LOADS WORD, HALF-WORD, OR BYTE WIDE PROGRAM TO * DIFFERENT SPECIFIED LOCATIONS. THE 8 LSBs OF THE FIRST MEMORY SPECIFIES THE * MEMORY WIDTH. IF THE HALF-WORD OR BYTE WIDE PROGRAM IS SELECTED, THE LSBs * ARE LOADED FIRST AND THEN THE MSBs. THE NEXT 2 WORDS CONTAIN THE CONTROL * WORD FOR THE GLOBAL AND LOCAL MEMORY INTERFACE CONTROL REGISTERS. NEXT COME * THE PROGRAM BLOCKS. THE FIRST TWO WORDS OF EACH PROGRAM BLOCK CONTAIN THE * BLOCK SIZE AND DESTINATION ADDRESS WHERE THE PROGRAM IS TO BE LOADED. WHEN * THE ZERO BLOCK SIZE IS READ, THE PROGRAM BLOCK LOADING IS TERMINATED. THE * NEXT TWO WORDS ARE THE INITIAL VALUES FOR THE IVTP AND TVTP REGISTERS. * AFTER THE BOOTLOADING IS COMPLETED, THE IACK SIGNAL IS SENT OUT ACCORDING TO THE LAST WORD OF THE SOURCE MEMORY, AND THE PROGRAM COUNTER WILL BRANCH * TO THE STARTING ADDRESS OF THE FIRST PROGRAM BLOCK. * NOTE 4.IF IIOF(3-0) ARE SET FOR COMMUNICATION PORT BOOTLOADER, THE PROCESSOR WAITS FOR THE FIRST INPUT FROM AN INPUT COMMUNICATION CHANNEL AND USE THAT * CHANNEL TO PERFORM THE DOWNLOAD. THE BEGINNING TWO WORDS SHOULD CONTAIN THE * GLOBAL AND LOCAL BUS CONTROL WORDS. SIMILAR TO THE MEMORY LOADER, THE * PROGRAM CAN BE LOADED INTO DIFFERENT MEMORY BLOCKS. THE FIRST TWO WORDS OF * EACH PROGRAM BLOCK CONTAINS THE BLOCK SIZE AND MEMORY ADDRESS TO BE LOADED * INTO. WHEN THE ZERO BLOCK SIZE IS READ, THE PROGRAM BLOCK LOADING IS * TERMINATED. IN OTHER WORDS, TO TERMINATE THE PROGRAM BLOCK LOADING, A * ZERO HAS TO BE ADDED AT THE END OF PROGRAM BLOCK. THE FOLLOWING TWO WORDS * ARE THE INITIAL VALUES FOR THE IVTP AND TVTP REGISTERS. AFTER THE BOOT-* LOADING IS COMPLETED, THE IACK SIGNAL IS SENT OUT ACCORDING TO THE LAST * WORD OF THE SOURCE MEMORY AND THE PROGRAM COUNTER BRANCHES TO THE STARTING * ADDRESS OF THE FIRST PROGRAM BLOCK.

10-20

	.sect	"boot"					
********	* * * * * * * *	* * * * * * * * * * * * * * * * * *	* * * * * * * * * * * * * * * * * * * *				
*	TM	S320C4x PROCESSOR	BOOTLOADER *				
*********	*******	********					
B001:	LDI	COM_LOAD, RIU	; Comm. port load subroutine address -> RIU				
*	LDHI	UUIUH, ARU	; Load peripheral mem. map start addr 100000H				
* CHECK	CHECK THE IIOF1-3 FOR THE BOOTLOADER						
*							
CHECK:	LDHI	0030H,AR1	; Load memory address = 00300000H				
	CMPI	04404H, IIF	; Test function 110 condition				
	BEQ	MEMORY	; If true, execute memory bootloader				
	LDHI	04000H, AR1	; Load memory address = 40000000H				
	CMPI	04044H, IIF	; Test function 101 condition				
	BEQ	MEMORY	; li true, execute memory bootloader				
	трит	060004 201	; Load momory addross = 6000000H				
	· 'C44	· LOHT 000805 AR1	· replace previous line with this line ('C44)				
	, 011	. 1011 000001,711(1	:				
	CMPI	04004H, IIF	; Test function 100 condition				
	BEQ	MEMORY	; If true, execute memory bootloader				
	LDHI	08000H,AR1	Load memory address = 80000000H				
	CMPI	00444H, IIF	; Test function 011 condition				
	BEQ	MEMORY	; If true, execute memory bootloader				
			;				
	LDHI	0A000H,AR1	; Load memory address = A0000000H				
	; 'C44	: LDHI 08040,AR1	; replace previous line with this line ('C44)				
	CMD T	0040411 110	; 				
	CMPI	VU4U4H, LIF	; lest function UIU condition				
	BFŐ	MEMORY	, II true, execute memory bootloader				
	T.DHT	OCOOOH AR1	. Load memory address = C000000H				
	: 'C44	: LDHI 08080H	: replace previous line with this line ('C44)				
	,		;				
	CMPI	00044H,IIF	; Test function 001 condition				
	BEQ	MEMORY	; If true, execute memory bootloader				
	CMPI	00004H,IIF	; Test function 000 condition				
	BEQ	STATRAM	; If true, branch to STATIC RAM TEST				
*		COMMUNICATIO	N PORT BOOTLOADER				
*							
*							
* CHECK	COMMUN	ICATION PORT INPU	I CHANNEL				
*							
	ADDI	040H, AR0, AR3	; Point to comm. port 0 control register addr				
	ĹDI	5,AR1	; Set loop counter for CHECK_CH loop				
CHECK_CH:	LSH3	-9,*AR3,R1	; Check comm port input				
	BNZ	LOADU	; If input exist, start comm port loader				
	ADDI	UIUH, AR3	; Point to next comm. port channel addr				
	DRO	AKI, CHECK_CH	; Uneck next comm. port channel input				
	Ы	CHECK	; Recheck the input flags				

The Bootloader Program

*		MEMOR	RY BOOTLOADER
* * TES	T MEMORY	WORD WIDTH	
MEMORY:	LDI LDI LSH BN	*AR1++(1),R1 W_WIDE,R10 26,R1 LOAD0	; Load the memory word width ; Full-word size subroutine address -> R10 ; Test bit5 of mem. width word ; If '1' start PGM loading (32 bits width)
	NOP LDI LSH BN	*AR1++(1) H_WIDE,R10 1,R1 LOAD0	; Jump last half word from mem. word ; Half-word size subroutine address -> R10 ; Test bit4 of mem. width word ; If '1' start PGM loading (16 bits width)
*	NOP LDI NOP	*AR1++(1) B_WIDE,R10 *AR1++(1)	; Jump last 1 bytes from mem. word ; Byte size subroutine address -> R10 ; Jump last 1 bytes from mem. word
* STA *	RT PROGRA	M LOADING	
LOAD0:	LAJU LDHI LDI NOP	R10 0010H,AR0 1,R0	; Load new word according to mem. width ; Load peripheral mem. map start addr 100000 ; Set start address flag off
	LAJU STI NOP NOP	R10 AR2,*AR0	; Load new word according to mem. width ; Set global bus control register
	STI	AR2,*+AR0(4)	; Set local bus control register
LOAD2:	LAJU ADDI NOP NOP	R10 1,R0	; Load new word according to mem. width ; Set start address flag off
	CMPI BEQ	0,AR2 IVTP_LOAD	; If 0 block size start PGM
	LAJU SUBI3 NOP	R10 1,AR2,RC	; Load new word according to mem. width ; Set block size for repeat loop
	SUBI LDI LDIP LAJU LDI LDI ADDI B	1,R10 R0,R0 AR2,R9 R10 AR2,AR0 -1,R0 1,R10 LOAD2	<pre>; Sub address with loop ; Test start address loaded flag ; Load start address if flag off ; Load block words according to mem. width ; Set destination address ; Set start & dest. address flag on ; Sub address without loop ; Jump to load a new block when loop complet</pre>

INITIALIZE IVTP AND TVTP REGISTERS IVTP_LOAD: LAJU R10 ; Load new word according to mem. width NOP NOP NOP TVTP_LOAD: LAJU R10 ; Load new word according to mem. width LDPE AR2, IVTP ; Load the IVTP pointer NOP NOP LAJU R10 ; Load new word according to mem. width LDPE AR2, TVTP ; Load the TVTP pointer NOP NOP ; Send out IACK signal out IACK *AR2 ; Branch to the start of the program BU R9 -----; BYTE-WIDE MEMORY BOOTLOADER SUBROUTINE ; ; -----; RPTB LOAD_B ; PGM load loop LOOP_B: LWL0 *AR1++(1),AR2 ; Load byte 0 (LSB) B_WIDE: NOP ; Nop for STRB to go high LWL1 *AR1++(1), AR2 ; Join byte 1 with byte 0 NOP ; Nop for STRB to go high LWL2 *AR1++(1),AR2 ; Join byte 2 with byte 0 & 1 ; Nop for STRB to go high NOP LWL3 *AR1++(1),AR2 ; Join byte 3 with byte 0, 1, & 2 LDI R0,R0 ; Test load address flag BNN B_END LOAD_B: SII BU STI AR2,*AR0++(1) ; Store new word to dest. address BU R11 ; Return from subroutine ; Return from subroutine -----; ; HALF-WORD WIDE MEMORY BOOTLOADER SUBROUTINE ;-----; LOOP_H: RPTB LOAD_H ; PGM load loop LWL0 *AR1++(1),AR2 ; Load LSB half-word H_WIDE: NOP ; Nop for STRB to go high LWL2 *AR1++(1),AR2 ; Join MSB half-word with LSB half-word ; Test load address flag LDI R0,R0 BNN H_END LOAD_H STI H_END BU AR2, *AR0++(1) ; Store new word to dest. address R11 ; Return from subroutine ;-----; ; FULL-WORD WIDE MEMORY BOOTLOADER SUBROUTINE ; ;-----; LOOP_W RPTB LOAD_W ; PGM load loop LDI *AR1++(1),AR2 ; Read a new 32 bits word W_WIDE LDI R0,R0 ; Test load address flag BNNW_ENDLOAD_WSTIAR2,*AR0++(1); Store new word to dest. addressW_ENDBUR11; Return from subroutine

The Bootloader Program

;	COMMUN	NICATION PORT BC	OTLOADER SUBROUTINE ;
LOOP_C COM_LOAD	RPTB LSH3 BZ LDI LDI BNN	LOAD_C -9,*AR3,R1 COM_LOAD *+AR3(1),AR2 R0,R0 C_END	; PGM load loop ; Check comm port input ; Wait for comm port input ; Read a new 32 bits word ; Test load address flag
LOAD_C C_END	STI BU .end	AR2,*AR0++(1) R11	; Store new word to dest. address ; Return from subroutine

Chapter 11

Page

The DMA Coprocessor

The direct memory access (DMA) coprocessor is a programmable on-chip device that allows simultaneous memory transfer and CPU operation with minimum CPU overhead. This chapter describes the DMA coprocessor and also offers suggestions for programming the device.

Topic

11.1Introduction11-211.2DMA Functional Description11-311.3DMA Registers11-711.4DMA Unified Mode11-1911.5DMA Split Mode11-2011.6DMA Internal Priority Schemes11-2211.7CPU and DMA Coprocessor Arbitration11-2711.8Data Transfer Modes11-2811.9Autoinitialization11-3411.10DMA and Interrupts11-4211.11DMA Memory Transfer Timing11-51

11.1 Introduction

The DMA coprocessor is a self programmable peripheral that transfers blocks of data by maximizing sustained CPU performance and by alleviating the CPU of burdensome I/O duties.

- Transfers to and from anywhere in the processor's memory map. For example, transfers can be made to and from on-chip memory, off-chip memory, and any of the six on-chip communication ports.
- Six DMA channels for memory-to-memory transfers in unified mode; a special split mode supports 12 DMA channels for communication port to/ from memory transfers.
- Automatic initialization of registers via linked lists stored in memory, allowing the DMA to run continuously without intervention by the CPU.
- Concurrent CPU and DMA coprocessor operation with DMA transfers at the same rate as the CPU (supported by separate internal DMA address and data buses)
- Source and destination address registers with variable indices, making it possible to step through matrices by row or column
- Bit-reversed addressing for FFTs
- Synchronization of data transfers via external and internal interrupts

11.2 DMA Functional Description

The DMA coprocessor supports six DMA channels that perform transfers to and from anywhere in the 'C4x memory map.

Each DMA channel is controlled by nine registers that are mapped in the 'C4x peripheral address space, as shown in Figure 11–1. The major DMA registers are described in Section 11.3.

The DMA coprocessor has dedicated on-chip address and data buses (see Figure 2–8 for a block diagram of the peripherals of the 'C4x). All accesses made by the six DMA channels are arbitrated in the DMA coprocessor and take place over these dedicated buses. The six DMA channels transfer data in a sequential time-slice fashion, rather than simultaneously, because they share common buses.

The DMA channels can run constantly or can be triggered by external (IIOF3–0) or internal (on-chip timers and communication ports) interrupts.

The DMA coprocessor can transfer data in a bit-reversed fashion (for FFT applications) or in a linear fashion; it can also transfer matrix data in a row or column fashion.

The DMA coprocessor has two basic operational modes:

- ❑ Unified Mode: Used for memory-to-memory transfers. The unified mode is described in Section 11.4, DMA Unified Mode. The unified block transfer sequence is presented in subsection 11.2.1, Block Transfer Sequence.
- **Split Mode**: Used for two-way, memory-to-communication port transfers. The split mode is described in Section 11.5, *DMA Split Mode*.



Figure 11–1. DMA Coprocessor Memory Map

Exploded View Of Each Channel Register

		_
010 00z0h	Control Registers x	1 ↑
010 00z1h	Source Address x	
010 00z2h	Source Address Index x	
010 00z3h	Transfer Counter x	DMA
010 00z4h	Destination Address x	Ch.
010 00z5h	Destination Address Index x	x
010 00z6h	Link Pointer x	
010 00z7h	Auxiliary Transfer Counter x	
010 00z8h	Auxiliary Link Pointer x	1

x = channel number (e.g., a 1 for all registers in channel 1, a 2 for all registers in channel 2, etc.).

z = corresponding hexadecimal digit for channel address (e.g., substitute an "A" for DMA channel 0; "B" for DMA channel 1, etc.).

11.2.1 DMA Basic Operation

If a block of data is to be transferred from one region in memory to another region in memory (unified mode), the following sequence is performed:

DMA Registers Initialization

- The source address register of a DMA channel is loaded with the address of the memory location to read from.
- 2) The destination address register of the same DMA channel is loaded with the address of the memory location to write to.
- 3) The transfer counter is loaded with the number of words to be transferred.
- 4) The source/destination index register is loaded with the step size of source/destination register update. If sequential memory accesses are required, the source address index register and the destination address index register must be set to 1.
- 5) The DMA channel control register is loaded with the appropriate modes to synchronize the DMA coprocessor reads and writes with interrupts. The DIE register determines which interrupt to use for synchronous transfer.

DMA Start

 The DMA coprocessor is started via the DMA START field in the DMA channel control register.

Word Transfers

- The DMA channel reads a word from the source address register and writes it to a temporary register within the DMA channel.
- 8) After a read by the DMA channel, the source-index register is added to the source address register.
- After the read operation completes, the DMA channel writes the temporary register value to the destination address pointed to by the destination address register.
- After the destination address has been fetched, the transfer counter register is decremented and the destination-index register is added to the destination-address register.

Note:

Both of the index registers (source and destination) contain signed values. This allows for variable step sizes or continuous reads from and/or writes to memory. When an index register equals zero, the DMA coprocessor transfers data to or from a fixed location.

11) During every data write, the transfer counter is decremented. The block transfer terminates when the transfer counter reaches zero and the write of the last transfer is completed. The DMA channel sets the transfer counter interrupt (TCINT) flag in the DMA channel control register.

After the completion of a block transfer, the DMA coprocessor can be programmed to do several things:

- Stop until reprogrammed (TRANSFER MODE bits = 01_2)
- **Continue transferring data (TRANSFER MODE bits = 00_2)**
- Generate an interrupt to signal the CPU that the block transfer is complete (TCC bit = 1₂)
- Autoinitialize itself to start the next block transfer (TRANSFER MODE bits = 10₂ or 11₂).

Each DMA channel reads new DMA register values from memory, loads these values into its register file, and, according to the values loaded, begins another block transfer. Whether or not the CPU must initialize transfers is determined by the value of the transfer mode bits:

- Autoinitialization under transfer mode bits = 10₂ is done without any intervention by the CPU.
- Autoinitialization under transfer mode bits =11₂ requires the CPU to start the DMA.

11.3 DMA Registers

Each DMA channel has nine registers designated as follows:

- Control register: contains the status and mode information about the associated DMA channel.
- Source address register: contains the memory address of data to be read.
- Source address-index register: contains the step size (a signed 32-bit number) used to increment or decrement the source address register.
- Destination address register: contains the memory address where data is written.
- Destination address-index register: contains the step size (a signed 32-bit number) used to increment or decrement the destination address register.
- Transfer counter register: contains the block size to move in unified mode or in split mode (primary channel).
- Auxiliary transfer-counter register: contains the block size to move in split mode (auxiliary channel).
- Link pointer register: contains the memory address of data to autoinitialize the DMA channel registers. Used for unified mode or primary channel in split mode.
- Auxiliary link-pointer register: contains the memory address of data to autoinitialize the DMA channel registers. Used for auxiliary channel in split mode.

After reset, the control register, the transfer counter, and the auxiliary transfer counter registers are set to zeros and the other registers are undefined.

11.3.1 Control Register

The format of the DMA-channel control register is shown in Figure 11–2. The text following the figure describes the functions of each field in the register.

At reset, each DMA-channel control register is set to zero. This makes the DMA channels lower-priority than the CPU, sets up the source address and destination address to be calculated via linear addressing, and configures the DMA channel in the unified mode.

31	30	29	28	27 26 2		25	:	24 23		2	22 21		21	20		19	Э	18
xx	PRIORITY MODE	AU STAT	AUX STATUS STAT		US AU		AUX START		ST	START		AUX TCINT FLAG		TCINT FLAG		AUX T	°CC	тсс
	RW	R	R F		7	RW		V-A		RW I		R		R	RWS	A	RWS	
		1	17	16		15		14	•		13		12		1	1	1	0
				COM	POR	Т	0 K	SPL Moi	.IT DE	WR	ITE B REV	BIT	READ B REV	IT	UX Al S۱	JTOINI ⁻ ′NC	r au	TOINIT SYNC
		RW	RW-A RW-A			RW-A R		RWSA		RWSA		RWS		RWSA		F	RWS	
			9		8		7	7		6	5		4		3	2	1	0
		AU	AUX AUTOINIT		AUTO ST/			SYNC MO		DDE AUX		X TI N	TRANSFER MODE		TRANSFER MODE		DMA PRI	
			RWSA		RWS		RW	/SA	R	WS	RW	SA	RWS	A	RWS	RWS	RWS	RWS

Figure 11–2. DMA Channel Control Register

R - Bit may be read.
 W - Bit may be written.
 S - Bit is shadowed during autoinitialization (no changes take place until autoinitialization is complete.)
 A - Bit is auxiliary for autoinitialization.
 xx - Reserved.

DMA Channel 0 only

DMA PRI	Sets DMA coprocessor priority. Defines the arbitration rules to be used when a DMA channel and the CPU are requesting the same resource. Affects all DMA coprocessor modes. The rules are listed in Table $11-1$.
TRANSFER MODE	Defines the transfer mode used by the DMA channel. Affects unified mode and the primary channel in split mode. The bits are defined in Table 11–2.
AUX TRANSFER MODE	Defines the transfer mode used by the DMA channel. Affects the <i>auxiliary</i> channel in split mode only. The bits are defined in Table 11–2.
SYNC MODE	Determines the mode of synchronization for performing data transfers, These bits work differently in unified and split modes. See Table 11–3 and Table 11–4 for bit descriptions for unified and split modes.
	Note: If a DMA channel is interrupt driven for both reads and writes, <i>and</i> the interrupt for the write comes before the interrupt for the read, the interrupt for the write is latched by the DMA channel. After the read is complete, the write can be executed.
AUTOINIT STATIC	This bit affects unified mode and the primary channel in split mode. It keeps the auxiliary link pointer constant during autoinitialization from the on-chip communication ports or other stream-oriented devices (such as first-in first-out (FIFO) memory buffers). <i>If bit=0</i> , the link pointer is incremented during autoinitialization. <i>If bit=1</i> , the link pointer is <i>not</i> incremented (it is static) during autoinitialization.

AUX Acts A AUTOINIT chant STATIC		Acts like the AUTOINIT STATIC bit above, except that it affects the auxiliary channel in split mode only.
	AUTOINIT SYNC	This bit has an effect only in the DMA coprocessor sync mode (bits 6–7 above). It affects the interrupt that is enabled by the DMA interrupt enable register (shown in Figure 11–25) used for DMA reads: <i>If bit</i> = 0, the interrupt is ignored, and the autoinitialization reads are not synchronized with any interrupt signals. <i>If bit</i> = 1, then the interrupt is recognized and is also used to synchronize the autoinitialization reads. This affects the unified mode and the primary channel in split mode (see the SPLIT MODE bit). The effect of this bit and the SYNC MODE bit in autoinitialization is summarized in Table 11–9.
	AUX AUTOINIT SYNC	Acts the same as the AUTOINIT SYNC bit above, except that it affects the auxiliary channel in split mode. The effect of this bit and the SYNC MODE bits in autoinitialization is summarized in Table 11–9.
READ BIT REV		Selects type of addressing for modifying the source address. <i>If bit=0</i> , the source address is modified using 32-bit linear addressing. <i>If bit = 1</i> , the source address is modified using 24-bit bit-reversed addressing. The bit affects unified mode and primary channel reads (source) in split mode.
	WRITE BIT REV	Selects the type of addressing for modifying the destination address. <i>If</i> $bit = 0$, the destination address is modified using 32-bit linear addressing. <i>If</i> $bit=1$, the destination address is modified using 24-bit bit-reversed addressing. The bit affects unified mode and auxiliary channel writes (destination) in split mode.
	SPLIT MODE	This bit controls the DMA coprocessor mode of operation. <i>If bit</i> = 0, DMA transfers are from memory to memory. This is referred to as <i>unified mode</i> . <i>If bit</i> = 1, <i>split mode</i> is entered with each DMA channel split into two channels, allowing a single DMA channel to perform memory-to-communication-port and communication-port-to-memory transfers. The split mode can be modified by autoinitialization in unified mode or by autoinitialization by the auxiliary channel in split mode. Split mode is further described in Section 11.4, <i>DMA Split Mode</i> .
	COM PORT	These bits define a communication port $(000_2 \text{ to } 101_2)$ to be used for DMA transfers. <i>If SPLIT MODE = 0</i> , COM PORT has no affect on the operation of the DMA channel. <i>If SPLIT MODE = 1</i> , COM PORT defines which of the six communication ports to use with the DMA channel. The COM PORT may be modified by autoinitialization in unified mode or by autoinitialization by the auxiliary channel in split mode.

TCC Transfer counter interrupt control. If TCC = 1, a DMA channel interrupt pulse is sent to the CPU after the transfer counter makes a transition to zero and the write of the last transfer is complete.

If enabled, the corresponding DMA interrupt (DMA INT0–INT5) occurs at the vector shown in Figure 7–2. *If TCC = 0*, a DMA channel interrupt pulse is not sent to the CPU when the transfer counter transitions to zero. This bit affects unified mode and the primary channel in split mode.

- **AUX TCC** Auxiliary transfer counter interrupt control. If bit = 1, a DMA channel interrupt pulse is sent to the CPU after the auxiliary transfer counter makes a transition to zero and the write of the last transfer is complete. If enabled, the corresponding DMA interrupt (DMA INT0–INT5) occurs as shown in Figure 7–2. If bit = 0, a DMA channel interrupt pulse is not sent to the CPU when the auxiliary transfer counter transitions to zero. This bit affects the auxiliary channel in split mode only.
- **TCINT FLAG** Transfer counter interrupt flag. This flag is set to 1 whenever the transfer counter makes a transition to zero and the write of the last transfer is completed. Whenever the DMA channel control register is read, this flag is cleared, *unless* the flag is being set by the DMA in the same cycle as the read. The TCINT FLAG is affected by the unified mode and the primary channel in split mode.
- AUX TCINT FLAG Auxiliary transfer counter interrupt flag. This flag is set to 1 whenever the auxiliary transfer counter makes a transition to zero and the write of the last transfer is completed. Whenever the DMA control register is read, this flag is cleared, *unless* the flag is being set by the DMA coprocessor in the same cycle as the read. The AUX TCINT FLAG is affected by the auxiliary channel in split mode. Since only one interrupt is available for a DMA channel, you can determine what event had set the interrupt by examining the TCINT FLAG and the AUX TCINT FLAG.
- **START** Starts and stops the DMA channel in several different ways (as are listed in Table 11–5). START affects the unified mode and the primary channel in split mode. If they is used to hold a channel in the middle of an autoinit sequence, the START and AUX START bits will hold the autoinit sequence. If the START or AUX START bits are being modified by the DMA channel (for example, to force a halt code of 10₂ on a transfer-counter terminated block transfer) and a write is being performed by an external source to the DMA channel control register, internal modification of the START or AUX START bits by the DMA channel has priority. See TRANSFER MODE bits value of 01₂ in Table 11–2 for more information.
- **AUX START** Starts and stops the DMA channel in several different ways (as are listed in Table 11–5). AUX START affects the auxiliary channel in split mode only.

STATUS	Indicates the status of the DMA channel as listed in Table 11–6. STATUS is updated in the unified mode and by the primary channel in the split mode. Updates are performed every cycle. The STATUS and AUX STATUS bits also determine if the DMA channel has halted or has been reset after writing to the START or AUX START bits.
AUX STATUS	Indicates the status of the DMA channel as listed in Table 11–6. STATUS is updated by the auxiliary channel in split mode only. Updates are performed every cycle.
PRIORITY	Priority mode of DMA channel access: If $bit = 0$, priority rotates as shown in

MODE Section 11.6. If bit = 1 priority is fixed as shown in Section 11.6. This bit is available only at DMA channel zero.

DMA PRI Bit Nos:	
1 – 0	Description
0 0	DMA coprocessor access is <i>lower</i> priority than CPU access. If the DMA channel and the CPU are requesting the same resource, then the CPU will proceed. These bits are set this way at reset.
0 1	This setting selects <i>rotating arbitration</i> , which sets priorities between the CPU and DMA channel by alternating their accesses, but not exactly equally. Priority rotates between CPU and DMA accesses when they conflict during consecutive instruction cycles. The first time the DMA channel and the CPU request the same resource, the CPU has priority. If, in the following instruction cycle, the DMA coprocessor and the CPU again request the same resource, the DMA has priority. Alternate access continues as long as the CPU and DMA requests conflict in consecutive instruction cycles. When there is no conflict in a previous instruction cycle, the CPU has priority.
10	Reserved.
11	DMA coprocessor access is <i>higher</i> priority than CPU access. If the DMA channel and the CPU are requesting the same resource, then the DMA will proceed.

Table 11–1. DMA PRI Bits and CPU/DMA Arbitration Rules

Table 11 2 TRANSEED MODE	ALLY TOANGEED MODE) Field Decorintions
TADIE TT-2. TRANSFER WODE) Field Descriptions

TRANSFER MODE Bit Nos:	Description
3 - 2 / (5 - 4)	Description
0 0	Transfers are <i>not</i> terminated by the transfer counter, and <i>no</i> autoinitialization is performed. TCINT (transfer counter interrupt) and AUX TCINT can still be used to cause an interrupt when the transfer counter makes a transition to zero. The DMA channel continues to run. Note that the address continues to increment while the transfer count rolls over to its maximum value of 0FFFF FFFFh.
0 1	Transfers are terminated by the transfer counter. No autoinitialization is performed. A halt code of 10 ₂ is placed in the START (or AUX START) field when transfers are completed.
10	Autoinitialization is performed when the transfer counter goes to zero without waiting for CPU intervention.
1 1	The DMA channel is autoinitialized when the CPU restarts the DMA coprocessor by using the DMA register in the CPU. When the transfer counter goes to zero, operation is halted until the CPU starts the DMA coprocessor by using the START (AUX START) field in the DMA channel control register (bits 22–23 and 24–25, Table 11–5). A halt code of 10_2 is placed in the START (or AUX START) field by the DMA coprocessor.

SYNC MODE Bit Nos:	
7 – 6	Description
0 0	No synchronization. Interrupts are ignored, see Figure 11–27.
0 1	Source synchronization. A read is not performed until an enabled interrupt occurs (see Figure 11–28a). The interrupt is specified by the DMAx READ field of the DMA interrupt enable (DIE) register (see subsection 11.10.1, <i>Interrupts and Synchronization of DMA Channels</i> , for more information).
10	Destination synchronization. A write is not performed until an enabled interrupt oc- curs (see Figure 11–29a). The interrupt is specified by the DMAx WRITE field of the DMA interrupt enable (DIE) register (subsection 11.10.1, <i>Interrupts and Synchroniza- tion of DMA Channels</i> , for more information).
11	Source and destination synchronization. A read is performed when an enabled inter- rupt (specified by the DMAx READ field) occurs. Then, a write is performed when an enabled interrupt (specified by the DMAx WRITE field) occurs (as shown in Figure 11–30). These fields are part of the DMA interrupt enable (DIE) register (see subsection 11.10.1, <i>Interrupts and Synchronization of DMA Channels</i> , for more in- formation).

	Table 11-3.	. SYNC MODE	Field Descr	iptions in	Unified	Mode
--	-------------	-------------	-------------	------------	---------	------

Table 11–4. SYNC MODE Field Descriptions in Split	Mode
---	------

SYNC MODE Bit Nos:	
7-6	Description
0 0	No synchronization. Interrupts are ignored see Figure 11–27.
0 1	Destination synchronization. A primary channel write to the communication-port out- put FIFO is not performed until an enabled interrupt occurs (see Figure 11–29b). The interrupt is specified by the DMAx PRIMARY WRITE field of the DMA interrupt en- able (DIE) register (see subsection 11.10.1, <i>Interrupts and Synchronization of DMA</i> <i>Channels</i> , for more information).
10	Source synchronization. An auxiliary-channel read from the communication-port in- put FIFO is not performed until an enabled interrupt occurs (see Figure 11–28b). The interrupt is specified by the DMAx AUXILIARY READ field of the DMA interrupt en- able (DIE) register (see subsection 11.10.1, <i>Interrupts and Synchronization of DMA</i> <i>Channels</i> , for more information).
11	Source and destination synchronization. A read from the communication-port input FIFO is performed when an enabled interrupt (specified by the DMAx AUXILIARY READ field) occurs. A write to the communication port output FIFO is performed when an enabled interrupt (specified by the DMAx PRIMARY WRITE field) occurs. These fields are part of the DMA interrupt enable (DIE) register (see subsection 11.10.1, <i>Interrupts and Synchronization of DMA Channels</i> , for more information).

START (AUX START) Bit Nos: 23 – 22	
(25 – 24)	Description
0 0	DMA channel reset. DMA-channel read or write cycles in progress are completed (not aborted); any data read is ignored. Any pending (not started) read or write is canceled. The auxiliary (AUX START =00 ₂) and primary (START=00 ₂) transfer counters are set to zero. The DMA channel is reset so that when it starts, a new transaction begins; that is, a read is performed. In this mode, stopping is immediate with no other registers loaded.
0 1	DMA halt on read or write boundary. Halts the DMA channel on the first available read or write boundary. If a read or write has begun, the read or write is completed before stopping. If a read or write has not begun, no read or write is started. In this mode, stopping is immediate with no other registers loaded).
1 0	DMA halt on transfer boundary. Halts the DMA channel on the first available transfer boundary. If a DMA transfer has begun, the entire transfer is completed, including both cycles (both read and write operations), before stopping. If a transfer has not begun, none is started. In this mode, stopping is immediate with no other registers loaded. This is also the value after a DMA transfer completes.
1 1	DMA start. Writing 11_2 to this field starts the DMA process using the values in the channel's DMA channel registers (Figure 11–1). If the DMA is in autoinitialization, all DMA registers are loaded before starting the operation. The DMA coprocessor starts from reset if previously reset (START or AUX START bits = 00_2) or restarts from the previous state if previously halted (START or AUX START bits = 01_2 or 10_2).

Table 11–5. START (AUX START) Field Descriptions

Table 11–6. STATUS (AUX STATUS) Field Descriptions

STATUS (AUX STATUS) Bit Nos: 27 – 26 (29 – 28)	Description
0 0	The DMA channel is held on the boundary of the DMA transfer (the write is com- plete, and the read has not begun). This is the value at RESET after a halt on a transfer boundary or after a block transfer.
0 1	The DMA channel is being held in the middle of a DMA transfer; (the read is complete, and the write has not begun). This occurs only if the START (or AUX START) field = 01_2 .
1 0	Reserved.
11	The DMA channel is not being held or reset.

11.3.2 Address and Index Registers

As shown in Figure 11–3, both the DMA coprocessor source-address and destination-address registers have an associated index register. After each DMAchannel read (source address) or write (destination address), the corresponding (source or destination) address generator adds the index register to the address register *and places the result in the address register*. In this way, the address register acts as an accumulator because it retains its own sum and the sum of its index register, as is shown by the following equation:

Address Register + Index Register \rightarrow Address Register

The values in these registers are undefined at reset.

Depending upon bits 12 and 13 (READ BIT REV and WRITE BIT REV) of the DMA channel control register, the addition may be either:

- Linear (normal addition): READ BIT REV = 0 or WRITE BIT REV = 0, or
- Bit reversed (reverse carry propagation): READ BIT REV = 1 or WRITE BIT REV = 1.

Both index values (source or destination) are *signed* values.



Figure 11–3. DMA Coprocessor Address Generation

(b) Destination address register operation

11.3.3 Transfer Counter and Auxiliary Transfer Counter Registers

These registers contain the number of words to be transmitted.

Figure 11–4 shows the six transfer counters and the six auxiliary transfer counters. A DMA channel in split mode (described in Section 11.4, *DMA Split Mode*) uses the auxiliary transfer counter for the auxiliary channel and the primary transfer counter for the primary channel. The values in these registers are set to zero at reset.

The counters are decremented after completing the address fetch for the write portion of a transfer. The TCINT FLAG and AUX TCINT FLAG (bits 20 and 21

of the DMA channel control register, as shown in Figure 11–2) are not set *until* the counter is decremented and the write of the last transfer is completed. Correspondingly, the interrupt will not be seen by the CPU interrupt controller until the transfer counter is decremented and the write of the last transfer is completed.

The decrementer checks whether the transfer counter equals zero after the decrement is performed. As a result, if the counter register has a value of 1, then the DMA channel can be halted after only one transfer is performed. Thus, by setting the transfer counter to 1, the DMA channel transfers the minimum possible number of words (1 time). The count is treated as an unsigned integer. Transfers can be halted when a zero count is detected after a decrement. If the DMA coprocessor channel is not halted after the transfer reaches zero, the counter will continue decrementing below zero. Thus, by setting the transfer counter to zero, the DMA channel transfers the maximum possible number of words (10000 0000h times).

Figure 11–4. Transfer Counter Registers



† x = DMA channel number (0-5)

11.3.4 Link Pointer and Auxiliary Link-Pointer Registers

The link pointers specify the address from which to load the new DMA channel register values when autoinitialization is performed. When a channel has exhausted its counter (transfer counter = 0), it will (if appropriately configured) use the link pointer to reload itself. Figure 11–5 illustrates the DMA coprocessor link address registers. The values in these registers are undefined at reset.

For example, under autoinitialization, the steps to load the channel registers for DMA channel 0 (as shown in Figure 11-1) are:

- Get the link pointer for the next DMA operation. The pointer is the memory address containing the contents of the first DMA channel 0 register (the channel control register as shown in Figure 11–1).
- Bring in the contents pointed to by the pointer and write to address 0010 00A0h (first word of DMA channel 0 registers as shown in Figure 11–1).
- 3) Increment the link pointer. (Skip this step if the AUTOINIT STATIC bit = 1.)
- 4) Bring in the next word and write to address 0010 00A1h.
- 5) Repeat until the entire block of registers is loaded for DMA channel 0 (7 registers in unified mode; 5 registers in split mode).

Figure 11–5. Link Pointer Registers



† x = DMA channel number (0–5)

11.4 DMA Unified Mode

Unified mode is the default DMA operational mode. It is used for memory-tomemory transfers. To select unified mode, clear the SPLIT MODE bit (bit 14 of the DMA channel control register, which is shown in Figure 11–2). Thus, write a zero to this bit (zero is the reset value of this bit).

The block transfer sequence under unified mode is covered in subsection 11.2.1. DMA channel arbitration in unified mode is described in Section 11.6. DMA synchronization with interrupts is covered in Section 11.10, *DMA and Interrupts*. Autoinitialization in unified mode is covered in subsection 11.9.1, *Unified Mode*.

A unified DMA word transfer consists of two steps, as shown in Figure 11–6:

- 1) The DMA channel reads the source data value from the address pointed to by the source address register and stores it in a temporary register.
- The DMA channel reads the temporary register value and writes it to the address pointed to by the destination address register.

You can use unified mode to perform communication port transfers, especially unidirectional transfers. Using split mode is more advantageous in bidirectional transfers.

Figure 11–6. Typical Unified-Mode DMA Channel Configuration



11.5 DMA Split Mode

The DMA split mode (see Figure 11–7) allows one DMA channel to be used for both reading and writing data to a communications port. Split mode essentially transforms one DMA channel into two DMA channels:

- Primary Channel: dedicated to reading data from a location in the memory map (external/internal) and writing it to a communication port output FIFO.
- Auxiliary Channel: dedicated to receiving data from a communication port input FIFO and writing it to a location in the memory map.

To select split mode, set the SPLIT MODE bit (bit 14 of the DMA channel control register, Figure 11–2) to one.

All six DMA channels support this split mode to accommodate all of the communication ports. The COM PORT field (bits 15–17 as shown in Figure 11–2) of the DMA channel control register defines which communication port is used (port 0–5). A DMA channel in split mode can be used with any communication port; however, read/write synchronization is restricted to signals from the communication port with the same number as the DMA channel being used; in other words, DMA*i* can synchronize only with signals coming from communication port *i* (see Section 11.10, *DMA* and Interrupts, for more information). Figure 11–7 shows typical split mode operation with one communication port.

A split mode word transfer is similar to that of the unified mode except for the following differences:

- The primary channel reads a word from the address pointed to by the source address register and writes it to a temporary register within the DMA coprocessor. It then writes the temporary register value to the output FIFO on the communication port specified in the COM PORT field. The registers that control the primary channel are the DMA channel control register, source address register, source index register (added to source address register), transfer-counter register, and link pointer register.
- The auxiliary channel reads a word from the input FIFO on the communication port specified in the COM PORT field and writes it to a temporary register within the DMA coprocessor. It then writes the temporary register value in the address pointed to by the destination address register. The registers that control the auxiliary channel are the DMA channel control register, destination address register, destination index register (added to the destination address register), auxiliary transfer-counter register, and auxiliary link pointer register.

DMA channel arbitration in split mode is described in subsection 11.6.3, *Split Mode and DMA Channel Arbitration*. DMA synchronization with interrupts is covered in Section 11.10, *DMA and Interrupts*. Autoinitialization in split mode is covered in subsection 11.9.2, *Split Mode*.

Figure 11–7. Typical Split-Mode DMA Configuration



Notice that there is only one temporary register in each DMA channel. Therefore, a primary channel operation must complete before an auxiliary channel operation can begin, and vice versa.

Primary and auxiliary channels share some of the DMA channel control registers and exclusively use others:

- PRIORITY MODE, COM PORT, SPLIT MODE, and DMA PRI are fields that both primary and auxiliary channels use.
- AUX STATUS, AUX START, AUX TCINT flag, AUX TCC, WRITE BIT REV, SYNC MODE (bit 7), and AUX TRANSFER MODE are used exclusively by the auxiliary channel.
- STATUS, START, TCINT flag, TCC, READ BIT REV, SYNC MODE (bit 6), and TRANSFER MODE are used exclusively by the primary channel.

11.6 DMA Internal Priority Schemes

Because all accesses made by the six DMA channels take place over one common internal DMA data and address bus, a priority scheme for bus arbitration is required. Within the DMA coprocessor, two priority schemes are used to designate which channel is serviced next:

- ☐ A fixed priority scheme with channel 0 always having the highest priority and channel 5 the lowest.
- A rotating priority scheme that places the most recently serviced channel at the bottom of the priority list (default setup after reset).

11.6.1 Fixed Priority Scheme

This scheme provides a fixed (unchanging) priority for each channel as follows:

Highest priority 0 1 2 3 4 Lowest priority 5

To select fixed priority, set the PRIORITY MODE bit (bit 30) of channel 0's DMA-channel control register to 1 (one).

11.6.2 Rotating Priority Scheme

In a rotating priority scheme, the last channel serviced becomes the lowest priority channel. The other channels sequentially rotate through the priority list with the lowest channel next to the last-serviced channel becoming the highest priority on the following request. The priority rotates every time the channel most recently granted priority completes its access. Figure 11–8 and Figure 11–10 illustrate the rotation of priority across several DMA coprocessor accesses. At system reset, the channels are ordered from highest to lowest priority (0, 1, 2, 3, 4, 5).

To select this scheme, set the PRIORITY MODE bit (bit 30) of **channel 0's** DMA control register to 0 (zero).

Figure 11–8. Rotating Priority Mode Example of the DMA Coprocessor



[†]DMA channel requesting an access

Each service is one read access or one write access. See Figure 11–9 for an example of a read/write sequence.

At the start of the example in Figure 11–8, channels 2, 4, and 5 are requesting service. Because channel 2 has the highest priority, it is serviced first. It then becomes the lowest priority channel. The highest priority channel then becomes channel three. On the following services, channels 4 and 5 are taken care of in a similar fashion. Figure 11–9 shows the entire read and write sequence.

Note:

Each service means one read access or one write access. The DMA coprocessor handles channel arbitration on an access-by-access basis; that is, a DMA channel must contend for both the read and the write access in both unified and split modes.

Figure 11–9. Rotating Priority DMA Read and Write Sequence Example (Unified Mode)

1st	2nd	3rd	4th	5th	6th	7th
Service	Service	Service	Service	Service	Service	Service
0 1 12 ← DM/ 3 14 15	3 +4 +5 0 1 +2	$ \begin{array}{cccc} & 15 & 4 & DM \\ & 1 & & R \\ & 1 & & 12 \\ & 1 & & 12 \\ & 3 & & 14 \\ & & 14 & & 14 \end{array} $	A 0 1 t2 ↓ 3 t4 t5	3 14 ₩ 0 1 +2	HA 0 ₩ 1 +2 3 +4	MA 0 W 1 t2 3 t4 t5

[†]DMA channel requesting an access

Figure 11–10 shows the same results in a different way as in Figure 11–8 in a rotating priority scheme. Priority decreases from highest to lowest in a clockwise direction. The priority rotates in a counter clockwise direction with the most recently serviced channel becoming the lowest in priority.
Figure 11–10. Example of a Priority Wheel



[†] DMA channel requesting an access

With the rotating priority scheme, any DMA channel requesting service is guaranteed to be recognized after a number of higher priority requests have been serviced. The maximum number of requests are:

- Five in unified mode
- Eleven in split mode

This provides a way of preventing a channel from monopolizing the system.

DMA channels that are running and are not synchronized via interrupts are always requesting service.

11.6.3 Split Mode and DMA Channel Arbitration

When a DMA channel is running in split mode, arbitration between channels is similar to rotating priorities. A split-mode DMA channel has the same priority as a unified DMA channel. The only issue is how to arbitrate between the primary split channel and the auxiliary split channel. The split channels alternate priorities via a rotating priority scheme.

When a DMA channel is in split mode and both paths are simultaneously started via the START and AUX START bits, the output (primary) channel has priority over the input (auxiliary) channel. Both the START and AUX START bits must be written at the same time in order to achieve this reset condition.

The priority scheme for split mode channels is slightly different from the scheme for unified mode channels:

- For unified channels, the priority changes after a read or a write.
- For the primary and auxiliary channels within a split channel, priority changes after a complete read and write. This is because there is only one temporary register for both DMA channels (primary and auxiliary) to store the read value.

Figure 11–11 shows two channels contending for the DMA bus: channel 2 (a split channel) and channel 4.

Figure 11–11. Example of a Channel Priority Scheme in Split Mode



The channel priority scheme in Figure 11–11 is shown sequentially in Figure 11–12. In words, the scheme follows eight steps:

- The first service is a request by the primary split channel of channel 2 (2pri). 2pri reads, and then channel 2 is moved to the lowest priority level, but 2pri remains the higher priority channel of channel 2.
- 2) On the second service, channel 4, now a higher priority than channel 2, reads its source address and becomes the lowest priority.
- 3) On the third service, the value read by 2pri is written to its destination address, and channel 2 is moved to the lowest priority level. Also, 2pri is moved to a lower priority than 2aux. Note that the split channel that just completed a read retains a higher priority than the other split channel until the data is written to the destination address.
- On the fourth service, the value read by channel 4 in service 2 is now written to its destination address and the channel becomes the lowest priority.



Figure 11–12. Service Sequence for Split Mode Priority Example

- 5) In the fifth service, 2aux is read and channel 2 becomes the lowest priority.
- 6) On the sixth service, channel 4 is read again, and it becomes the lowest priority.
- 7) On the seventh and eighth services, the 2aux and channel 4 values that were read in services 5 and 6 are now written to their destination addresses. After the channel is written, it assumes the lowest priority.
- In the ninth service, 2pri is read again as in the first service, and the read/ write cycle continues as begun in the first service.

11.7 CPU and DMA Coprocessor Arbitration

The DMA coprocessor transfers data on its own internal buses. Arbitration is necessary only when a resource conflict exists between the DMA coprocessor and the CPU. The arbitration causes no delay. When there is no conflict, the CPU and DMA coprocessor accesses proceed in parallel.

All arbitration between the CPU and the DMA coprocessor is on an access basis; that is,the DMA coprocessor must contend for read and write accesses in both unified and split modes. *DMA coprocessor internal memory access starts during H3* (See Section 8.4, *Clocking of Memory Accesses*, on page 8-19, for more information).

When the CPU and DMA coprocessor request the same resource, the DMA channel's DMA PRI bits (bits 0 and 1 of the channel control register) define the arbitration rules (as shown in Table 11–7). The CPU has higher priority than the DMA when DMA PRI= 00_2 ; it has lower priority than the DMA when DMA PRI = 11_2 . They rotate priority when DMA PRI = 01_2 .

Table 11–7. DMA PRI Bits and CPU/DMA Arbitration Rules

DMA PRI (Bits 1–0)	Description
00	DMA access is lower priority than the CPU access. If the DMA channel and the CPU are requesting the same resource, then the CPU will proceed. (DMA PRI bits are set to 00_2 at reset.)
01	This setting selects <i>rotating</i> arbitration, which sets priorities between the CPU and DMA channel by alternating their accesses, but not exactly equally. Priority rotates between CPU and DMA accesses when they conflict during <i>consecutive instruction cycles</i> . The first time the DMA channel and the CPU request the same resource, the CPU has priority. If, in the following instruction cycle, the DMA co- processor and the CPU again request the same resource, the DMA has priority. Alternate access continues as long as the CPU and DMA requests conflict in consecutive instruction cycles. When there is no conflict in a previous instruction cycle, the CPU has priority.
10	Reserved
11	DMA access is higher priority than the CPU access. If the DMA channel and the CPU are requesting the same resource, <i>the DMA will proceed</i> .

11.8 Data Transfer Modes

Each DMA channel can operate in four types of data transfer modes. These modes differ in:

- U Whether or not they use autoinitialization
- How they operate if autoinitialization is in effect or not

Table 11–8 and the following paragraphs describe these data transfers.

Table 11–8. TRANSFER MODE (AUX TRANSFER MODE) Field Descriptions

TRANSFER MODE (AUX TRANSFER MODE)	
Bits 3–2 (5–4)	Description
0 0 ₂	Transfers are not terminated by the transfer counter. No autoinitialization is per- formed. The TCINT (transfer count interrupt) bits can still be used to cause an interrupt when the transfer counter makes a transition to zero. The DMA channel continues to run.
0 1 ₂	Transfers are terminated by the transfer counter. No autoinitialization is per- formed. A halt code of 10_2 is placed in the START or AUX START field (bits 22–23 or bits 24–25 of the DMA channel control register) when transfers are com- plete.
1 0 ₂	Autoinitialization 1. Autoinitialization is performed when the transfer counter goes to zero without waiting for CPU intervention.
1 1 ₂	Autoinitialization 2. The DMA channel is autoinitialized when the CPU restarts the DMA coprocessor by using the DMA channel control register in the CPU. When the transfer counter goes to zero, operation is halted until the CPU starts the DMA coprocessor by using the START (or AUX START) field in the DMA channel control register. A halt code of 10_2 is placed in the START (or AUX START) field by the DMA.

11.8.1 Running in TRANSFER MODE = 002

When TRANSFER MODE = 00_2 , transfers are not terminated when the transfer counter goes to zero, and no autoinitialization is performed. Even though the transfer counter does not halt transfers, an interrupt can be generated on the transfer counter transition to zero, setting the TCINT FLAG bit to 1. If the DMA coprocessor channel is not halted after the transfer reaches zero, the counter will continue decrementing below zero.

11.8.2 Running in TRANSFER MODE = 012

When TRANSFER MODE = 01_2 , transfers are terminated when the transfer counter goes to zero, and no autoinitialization is performed. When the transfer counter goes to zero, the DMA channel is halted by forcing 10_2 into the START or AUX START field.

11.8.3 Running in TRANSFER MODE = 10₂ (Autoinitialization 1)

This transfer mode allows the DMA channel to run continuously, change pointers and synchronization by the autoinitialization procedure, and turn itself off. Two different autoinitialization methods are supported:

Autoinitialization method 1a always starts after a system reset, after a DMA channel is reset (00_2 written to the START or AUX START bits), or after a DMA channel halts (01_2 or 10_2 written to START or AUX START bits). To select transfer mode 10_2 (autoinitialization method 1a), follow the steps listed here and shown in Figure 11–13.

- Initialize the DMA control register to transfer mode 10₂, and reset or halt the DMA channel to be autoinitialized.
- 2) Initialize the transfer counter to 0 (resetting the DMA channel does this).
- 3) Initialize the DMA channel link pointer with the address where the autoinitialization values reside. No initialization of the other DMA channel registers is required, because they are automatically set up during the autoinitialization process.
- 4) Start the DMA channel by writing 11₂ to the START (or AUX START) bits.
- 5) The DMA channel performs the sequence, *autoinitialize and block transfer.*

Figure 11–13. DMA Channel Running in Transfer Mode 102 (Autoinitialization Method 1a)



Autoinitialization method 1b starts when the transfer counter is not zero. The DMA starts a regular DMA transfer and autoinitializes after this transfer completes (when the transfer counter becomes zero). To select transfer mode 10_2 (autoinitialization method 1b), follow the steps listed here and shown in Figure 11–14.

- 1) Initialize the DMA control register to transfer mode 10₂, and reset or halt the DMA channel for the first transfer operation.
- 2) Initialize all the other DMA channel registers (source address, destination address, transfer counter, etc.) according to the transfer operation desired. Note that the transfer counter now reflects the number of words to be transferred (normally a nonzero value) before the autoinitialization process.
- Initialize the DMA channel link pointer with the address where the autoinitialization values for subsequent transfer operations reside.
- 4) Start the DMA channel by writing 11₂ to the START (or AUX START) bits.
- 5) The DMA channel performs this sequence: *block transfer and autoinitialize* (reverse order of method 1a).

Note that if a DMA channel is programmed to perform n block transfers, autoinitialization method 1a requires n DMA autoinitialization values. Autoinitialization method 1b requires only n-1 autoinitialization values because the first transfer can be accomplished during the initial DMA transfer. This represents some memory saving, but successive identical DMA operations require extra CPU cycles to set the initial DMA registers values again.

Figure 11–14. DMA Channel Running in Transfer Mode 10₂ (Autoinitialization Method 1b)



11.8.4 Running in TRANSFER MODE = 11₂ (Autoinitialization 2)

This transfer mode, besides having all of the advantages of autoinitialization, allows the CPU to coordinate its operation very easily with the operation of the DMA channels. Two different autoinitialization methods are supported:

Autoinitialization method 2a always starts after a system reset, after a DMA channel reset (00_2 written to the START or AUX START bits), or after a channel halts (01_2 or 10_2 written to the START or AUX START bits). To select transfer mode 11_2 and use autoinitialization method 2a, follow the steps listed here and shown in Figure 11–15.

- Initialize the DMA control register to transfer mode 11₂ and reset or halt the DMA channel to be autoinitialized.
- 2) Initialize the transfer counter to 0 (resetting the DMA channel does this).
- 3) Initialize the DMA channel link pointer with the address where the autoinitialization values reside. No initialization of the other DMA channel registers is required, because they are automatically set up during the autoinitialization process.
- 4) Start the DMA channel by writing 11₂ to the START or AUX START bits.
- 5) The DMA channel autoinitializes itself and performs a block transfer.
- 6) When the transfer counter goes to zero, the DMA waits for the CPU to write a 11₂ to the START(or AUX START) field of the DMA channel control register and autoinitialize.
- 7) Repeat the sequence autoinitialize, transfer, and wait.
- 8) When the transfer counter goes to zero, you can halt the DMA channel by forcing 10₂ into the START (or AUX START) field.

Figure 11–15.DMA Channel Running in Transfer Mode 11₂ (Autoinitialization Method 2a)



Autoinitialization method 2b starts when the transfer counter is not zero. The DMA starts with a regular DMA transfer and autoinitializes after this transfer completes (when the transfer counter becomes zero). To select transfer mode 11_2 and use autoinitialization mode 2b, follow the steps listed here and shown in Figure 11-16.

- Initialize the DMA control register to transfer mode 11₂ and reset or halt the DMA channel for the first transfer operation.
- 2) Initialize the other DMA channel registers (source address, destination address, transfer counter, etc.) accordingly. Note that the transfer counter now reflects the number of words to be transferred (normally a nonzero value) before the autoinitialization process.
- Initialize the DMA channel link pointer with the address where the autoinitialization values for subsequent transfer operations reside.
- Start the DMA channel by writing 11₂ to the START(or AUX START) bits.
- 5) The DMA channel performs the initial block transfer. When the transfer counter goes to zero, the DMA waits for the CPU to write a 11₂ to the START or AUX START field of the DMA channel control register and auto-initialize.
- 6) Repeat the sequence transfer, wait, and autoinitialize,.

Note that if a DMA channel is programmed to perform *n* block transfers, using autoinitialization method 2a requires *n* DMA autoinitialization values. Autoinitialization method 2b requires only n-1 autoinitialization values because the first transfer can be accomplished during the initial DMA transfer. This represents some memory saving, but successive identical DMA operations require extra CPU cycles to set the initial DMA register values again.



Figure 11–16. DMA Channel Running in Transfer Mode 11₂ (Autoinitialization Method 2b)

11.9 Autoinitialization

Autoinitialization is a method for reloading a DMA channel register file when the transfer counter goes to zero. When the DMA channel is operating in autoinitialization mode, the link pointer register and auxiliary link pointer register are used to initialize the registers that control the operation of the DMA channel. These pointers are memory address locations for blocks of data that are to be loaded into the DMA register file, shown in Figure 11–1. Link pointers are covered in subsection 11.3.4, *Link Pointer and Auxiliary Link–Pointer Registers*.

Autoinitialization is a regular DMA block transfer operation in which the destination is the DMA coprocessor's register file. The DMA reads the value pointed to by the link pointer and writes it to the DMA register over the peripheral bus on the next available cycle. Consequently, autoinitialization read/write accesses are also subject to any normal CPU/DMA access conflict.

Autoinitialization can happen:

- □ Without CPU intervention when the TRANSFER MODE bits = 10_2 (autoinitialization 1). Refer to subsection 11.8.3, *Running in TRANSFER MODE* = 10_2 (Autoinitialization 1).
- With CPU intervention when the TRANSFER MODE bits = 11₂ (autoinitialization 2). In this case, the CPU should restart the DMA channel before the autoinitialization proceeds. Refer to subsection 11.8.4, *Running in TRANSFER MODE = 11₂* (*Autoinitialization 2*).
- Before any block transfer (autoinitialization method a). The DMA starts with the transfer counter at zero, then autoinitializes and performs a block transfer.
- After a block transfer (autoinitialization method b). The DMA starts with a regular block transfer, and, when the transfer counter register goes to zero, it autoinitializes.

Autoinitialization 1 or 2 can use methods a or b.

Autoinitialization depends on the DMA channel's current mode: split or unified mode. The mode of operation is controlled by the SPLIT MODE bit (bit 14 in Figure 11–2). When autoinitializing the DMA coprocessor, *do not change* the SPLIT MODE bit. This bit should be changed *only when the DMA coprocessor has been reset and halted* (see DMA START bit description in Table 11–5 for more information).

11.9.1 Unified Mode

If the DMA channel is running in unified mode (SPLIT MODE = 0), the link pointer is used and the DMA-channel registers are loaded in the following order:

- 1) DMA-channel control register
- 2) Source-address register
- 3) Source-address index register
- 4) Transfer-counter register
- 5) Destination-address register
- Destination-address index register
- 7) Link-pointer register

The storage of new values for these registers in memory is illustrated in Figure 11–17.

Figure 11–17. Store New Values of DMA Channel Registers in Memory (SPLIT MODE = 0)

Link pointer (+0) +1 Source address +2 Source address index +3 Transfer counter +4 Destination address +5 Destination address index +6 Link pointer

Map of New Register Values in Memory

11.9.2 Split Mode

If the DMA channel is running in split mode (SPLIT MODE = 1), then the autoinitialize sequence depends upon which counter has terminated.

If the transfer counter register has gone to zero with SPLIT MODE = 1, then the link-pointer register is used for autoinitialization. In this case, the DMA channel registers are loaded in the following order:

- 1) DMA-channel control register
- 2) Source-address register
- 3) Source-address index register
- 4) Transfer-counter register
- 5) Link-pointer register

The storage of the new values for these registers in memory is illustrated in Figure 11–18.



Figure 11–18.Store New Values of DMA Channel Registers in Memory (SPLIT MODE = 1 and Transfer Counter = 0)

If the auxiliary transfer counter register has gone to zero with SPLIT MODE=1, then the auxiliary link pointer register is used for autoinitialization. In this case, the DMA channel registers are loaded in the following order:

- 1) DMA channel control register
- 2) Destination address register
- 3) Destination address index register
- 4) Auxiliary transfer count register
- 5) Auxiliary link pointer register

The storage of the new values of these registers in memory is illustrated in Figure 11–19.

Figure 11–19.Store New Values of DMA Channel Registers in Memory (SPLIT MODE = 1 and Auxiliary Transfer Counter = 0)



11.9.3 Incrementing the Link Pointer

During autoinitialization, the link pointer can be incremented or held constant:

When the link pointer is incremented, the autoinitialization values are stored in sequential memory locations, and the link pointer or auxiliary link pointer is incremented in order to access each of these locations. When you autoinitialize the DMA channel from a stream-oriented device, such as the on-chip communication ports or external FIFOs, you should hold the link pointer constant.

This can be controlled by the AUTOINIT STATIC and the AUX AUTOINIT STATIC bits of the DMA control register as follows:

- In unified mode, the AUTOINIT STATIC bit controls the link pointer.
- In split mode, the AUTOINIT STATIC bit controls the link pointer (primary channel), and the AUX AUTOINIT STATIC controls the auxiliary linker pointer.

When the AUTOINIT STATIC (AUX AUTOINIT STATIC) bit is zero, the link pointer is incremented. When it is one, the link pointer is held constant.

11.9.4 Synchronization

Usually, autoinitialization data is stored in memory, and synchronization is not necessary. In some cases, you may wish to transfer autoinitialization data in the same way as in the synchronized data reads and writes.

Autoinitialization synchronization is a function of the:

- SYNC MODE bits (DMA channel control register bits 6 and 7) that control synchronization of data transfers, and
- AUTOINIT SYNC bits (DMA channel control register bits 10 and 11) that affect only autoinitialization synchronization.

If the SYNC MODE bits are not set to synchronize data transfers (i.e., if the preceding data transfer is not synchronized on interrupts), then the DMA channel autoinitialization sequence is not synchronized either. If the SYNC MODE bits are set to transfer data synchronously (if the preceding data transfer is synchronized), then the upcoming data channel autoinitialization sequence can be synchronized on reads or writes or both (depending on whether the DMA coprocessor is in unified or split mode) as shown in Table 11–9. Note that when both modes show "no sync" for a bit setting in the table, the DMA channel autoinitialization sequence is not synchronized on interrupts.

In unified mode, there is no write synchronization for autoinitialization operation, because the destination is the DMA register, which is always ready.

In split mode, bit 6 of the DMA control register controls the autoinitialization synchronization of the DMA primary channel, and bit 7 controls the autoinitialization synchronization of the DMA auxiliary channel. If primary channel autoinitialization synchronization is used, the DMA read of autoinitialization values from memory does not proceed until the interrupt specified in the DMAx primary write field in the DIE register is received.

If auxiliary channel autoinitialization synchronization is used, the DMA read of autoinitialization values from memory does not proceed until the interrupt specified in the DMAx auxiliary read field in the DIE register is received.

SYNC MODE	AUTOINIT SYNC		
Bit Numbers 7 – 6	Bit Numbers 11 – 10	Unified Mode	Split Mode
 0 0	0 0	No synchronization	No synchronization
0 0	0 1	No synchronization	No synchronization
0 0	1 0	No synchronization	No synchronization
0 0	1 1	No synchronization	No synchronization
0 1	0 0	No synchronization	No synchronization
0 1	0 1	Read	Primary channel
0 1	1 0	No synchronization	No synchronization
0 1	1 1	Read	Primary channel
1 0	0 0	No synchronization	No synchronization
1 0	0 1	No synchronization	No synchronization
1 0	1 0	No synchronization	Auxiliary channel
1 0	1 1	No synchronization	Auxiliary channel
1 1	0 0	No synchronization	No synchronization
1 1	0 1	Read	Primary channel
1 1	1 0	No synchronization	Auxiliary channel
1 1	11	Read	Auxiliary and primary chan- nels

Table 11–9. Effect of SYNC MODE and AUTOINIT MODE Bits in Autoinitialization

11.9.5 Effect on DMA Control Register Bits

In unified mode, all of the writable control register bits are affected by autoinitialization. These bits are labeled in Figure 11–20.

In split mode during autoinitialization of the primary DMA channel, the writable, nonauxiliary bits can be modified, but auxiliary bits are protected (as shown in Figure 11–21). In other words, only nonauxiliary bits are allowed to be modified by the CPU or DMA coprocessor. Also, if the auxiliary DMA channel is autoinitialized, the writable auxiliary bits can be modified, but nonauxiliary bits are protected. These bits are labeled in Figure 11–22.

Even though the shadowed bits (designated by *s* in Figure 11–20) are modified during autoinitialization, they do not have an effect until autoinitialization is complete. Unshadowed bits take effect immediately, affecting the autoinitialization sequence. In other words, at autoinitialization, new shadowed bit values are entered last after all registers are loaded (as specified by the link pointer).

Regardless of whether the DMA channel is running in unified mode or split mode, if the CPU or another external source writes to the DMA channel control register, this affects all writable bits, including the shadow bits.

Note:

If the CPU writes to the DMA control register during DMA autoinitialization, the CPU write takes effect after the autoinitialization sequence completes. Even though the autoinitialization operation on the DMA registers is not affected, the subsequent data transfer may be affected.

31	30	29	28	27	26	25	24	23	22	21	2	20	19	9	18
xx	xx	xx	xx	xx	xx	A 51	UX ART	STA	RT	хх)	кх	AUX 1	гсс	тсс
						R/W	R/W	R/W	R/W				S	6	S
			1	7	16	15	14		13	12		11		1	0
				CON	I PORT	S	SPLIT MODE	- WI	RITE BIT REV	READ E	ЗIТ	AUX / SY	AUTO NC	AUT S\	O INIT /NC
			R	/W	R/W	R/W	S		S	S		:	S		S
			9		8		7	6	5	4		3	2	1	0
			AUX AU STAT	JTO IC	AUTO STAT	INIT IC	SYNC N	IODE	AUX ⁻	TRANSFER MODE	-	TRANS MOI	SFER DE	DMA	A PRI
		_	S		S		S	S	S	5	3	S	S	S	S

Figure 11–20.DMA Channel Control Register Bits Modifiable by Autoinitialization in Unified Mode

s — These shadowed bits do not take effect until autoinitialization is complete.

xx — Write protected during autoinitialization.

31	30	29	28	27	26	25	24	23	2	2	21	2	20	19	9	18
хх	хх	xx	xx	хх	хх	хх	XX	ST	ART		XX)	xx	x	х	TCC
								R/W	R/W							S
			17		16	15	14		13		12		1	1	1	10
			хх		xx	хх	xx		XX		READ REV	BIT /	,	κx	AUT S	fo init Ync
											s					S
			9		8		7	6		5	4		3	2	1	0
			xx		AUTO I STATI	NIT C	xx	SYNC MOD	; E	xx	xx		TRANS MO	SFER DE	DN	IA PRI
		_			S			S					S	S	S	S

Figure 11–21.DMA Channel Control Register Bit Modifiable by Autoinitialization of the Primary Channel in Split Mode

s — These shadowed bits do not take effect until autoinitialization is complete.

xx — Write protected during primary channel autoinitialization.

Figure 11–22.DMA Channel Control Register Bits That Can Be Modified by Autoinitialization of the Auxiliary Channel in Split Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	9	18
xx	xx	хх	xx	xx	xx	AUX	START	xx	xx	xx	хх	AUX	тсс	xx
						R/W	R/W					S	6	
			1	7	16	15	14		13	12	1	1	10)
				CC	M POF	RT	SP MO	LIT W	RITE BIT REV	xx	AUX S`	AUTO	×	x
			R	/W	R/W	R/W	S	;	S			S		
			9		8		7	6	5	4	3	2	1	0
		Å	AUX AU STATIO	TO C	xx	:	SYNC MODE	xx	AU>	K TRANSFE MODE	R _{xx}	хх	xx	xx
			S				S		S	S				

s — These shadowed bits do not take effect until autoinitialization is complete.

xx — Write protected during auxiliary channel autoinitialization.

11.9.6 Consecutive Autoinitializations

For many applications, it is sufficient to autoinitialize the DMA channel with the same data each time. In this case, the new link-pointer value points to the start of the same block of data containing the new link pointer, as illustrated in Figure 11–23. This particular example assumes that the DMA channel is not running in split mode.

If you want, you can make the new link pointer point to a new set of register values, as illustrated in Figure 11–24. This can be continued to any level.

Figure 11–23. Self-Referential Link Pointer



Figure 11–24. Referring to a New Link Pointer



+5

+6

Destination address index Link pointer



11-41

The DMA Coprocessor

11.10 DMA and Interrupts

The DMA coprocessor uses interrupts in the following way:

- It can send interrupts to the CPU when a block transfer finishes. See the TCC and AUX TCC bits in Figure 11–2.
- It can receive interrupts from the external interrupt pins (IIOF3–0), the timers, or the communication port (ICRDY, OCRDY).

This section explains how the DMA receives interrupts. This process is called synchronization.

All of the interrupts that the DMA coprocessor can see are first received by the CPU interrupt controller. Edge-triggered interrupts are latched by the CPU in the appropriate interrupt flag register; level-triggered interrupts are not.

When an external interrupt ($\overline{IIOF3}$ –0) is used for DMA coprocessor transfer synchronization, the CPU is responsible for configuring external interrupts as edge- or level-triggered interrupts (as set in the FUNCx and TYPEx bits of the interrupt flag register (discussed in subsection 3.1.10, *IIOF Flag Register (IIF)*), on page 3-13.

Edge-triggered interrupts are timer interrupts, DMA interrupts, and external interrupts that are configured as edge-triggered interrupts. Detailed information on interrupts is provided in Section 7.4, *Interrupts*, on page 7-15, and Section 7.6, *DMA Interrupts*, on page 7-26. When the interrupt controller determines that an edge-triggered interrupt that a DMA channel is waiting on (DIE registers bits set) has been latched into the interrupt flag, the CPU clears the interrupt flag and sends an interrupt pulse to the DMA channel. The DMA channel latches the interrupt locally until it can service the interrupt. At that time, the latched interrupt is cleared by the DMA coprocessor for two cycles.

Level-triggered interrupts generated by communication ports and external interrupts that are configured as level-triggered interrupts are handled differently by the CPU interrupt controller. When the interrupt controller determines that a level-triggered interrupt that a DMA channel is waiting for (DIE register bits set) has been received, the CPU sends an interrupt pulse to the DMA channel. The DMA channel latches the interrupt locally until it can service the interrupt. At that time, the locally latched interrupt is cleared by the DMA coprocessor for two cycles.

The interrupt reset signal generated by the DMA coprocessor after a DMA interrupt is serviced has priority over the interrupt set signal. Thus, the interrupt signal will not be continuously set, even if the CPU is continuously sending the interrupt set signal. Therefore, when the DMA-set priority scheme is used and a higher priority DMA channel is driven by continuous interrupt signals, the lower priority DMA channel can be serviced in between the higher priority DMA services.

Unlike the 'C3x, the 'C4x DMA processor is not affected by processing the CPU interrupts, even when pipeline fetches are being halted. When interrupts are enabled in the DIE register, the interrupt is latched automatically by the CPU interrupt controller and saved for future DMA use. When a flag interrupt (timer, external interrupt) is latched, the IIF flag is cleared. Note that IIF flags are cleared when the CPU interrupt controller latches the interrupt, not when the DMA responds to it. Even if the DMA has not been started, the interrupt latch occurs, except when the start bits in the DMA control register have the reset value (00_2 in the START or AUX START bits). DMA reset clears the interrupt internal latch. To avoid losing previously received interrupts, it is recommended that you initialize DIE register after starting the DMA, when the DMA start bits have the value 11_2 . Note that when the DMA completes a transfer, the start (AUXSTART) bits are set to 10_2 . For this reason, the DMA will not miss any interrupt between transfers.

The DMA and the CPU can respond to the same interrupt if the CPU is not involved in any pipeline conflict or in any instruction that halts instruction fetching. Refer to subsection 7.4.1, *Interrupt Vector Table and Prioritization*, on page 7-15 for more details. It is also possible for different DMA channels (including auxiliary and primary channels) to respond to the same interrupt. If the same interrupt is selected for source and destination synchronization, both read and write cycles are enabled with a single incoming interrupt.

The internal circuitry of the 'C4x guarantees proper operation between a communication port that generates level-triggered interrupts and the DMA channel that is synchronizing with those level-triggered interrupts.

Note:

When you synchronize the DMA channels with external interrupts, it is better to configure the interrupt lines as edge-triggered interrupts to ensure that only one interrupt is recognized.

11.10.1 Interrupts and Synchronization of DMA Channels

You can use interrupts to synchronize DMA channel transfers. To set up the DMA for a synchronous data transfer mode requires two steps:

 Set the DMA SYNC MODE bits (bits 6,7) in the DMA channel control register to the value for the source, destination, or source and destination synchronization desired. See subsection 11.10.2, *Synchronization Mode Bits*, for more information. 2) Set the DIE register to enable the corresponding interrupt for the DMA transfer synchronization desired. Figure 11–25 and Figure 11–26 show the DIE register for the split and unified modes, respectively. Table 11–10 and Table 11–11 lists the different synchronization interrupts for unified mode, and Table 11–12 and Table 11–13 list them for split mode.

It is recommended that you initialize the DIE register after starting the DMA, when the start bits have the value 11_2 . This prevents losing previously received interrupts, which may occur if you enable the DIE register when the start bits are 00_2 (reset value).

Figure 11–25. DIE Register Bit Functions for DMA Unified Mode

31	30	29	28	27	26	25	24	23	22	21	20
0	DMA5 Writ	е	D	MA5 Rea	ıd		0MA4 Writ	е	C	MA4 Rea	ıd
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
19	18	17	16	15	14	13	12	11	10	9	8
0	DMA3 Writ	е	D	MA3 Rea	ıd		0MA2 Writ	е	C	MA2 Rea	ıd
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
7		6	5		4	3		2	1		0
	DMA1 Writ	е	D	MA1 Rea	d	D	MA0 Write	e	D	MA0 Rea	d
R/W		R/W	R/W		R/W	R/W		R/W	R/W		R/W

R = Read W = Write

Table 11–10. DMA Channels 0 and 1 (DMA0 and DMA1) Unified-Mode Synchronization Interrupts

Bit Value	Inter	rupt Enabled	l at DMA0 o	r DMA1	
or DMA1)	DMA0 Read	DMA0 Write	DMA1 Read	DMA1 Write	Interrupt Source for DMA Synchronization
0 0†	None	None	None	None	
01	ICRDY0	OCRDY0	ICRDY1	OCRDY1	From communication port
10	IIOF0	IIOF1	IIOF2	IIOF3	From external pins IIOF0-IIOF3
11	TIM0	TIM0	TIM0	TIM0	From timer TIM0

[†] DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

Bit Value	Interrupt Enab	led at DMA2–DMA5 [†]	_ Interrupt Source for DMA
(in DMA2 to DMA5)	DMA <i>x</i> Read [†]	DMA <i>x</i> Write [†]	Synchronization
0 0 0‡	None	None	
001	ICRDY <i>x</i> †	OCRDY <i>x</i> †	From communication port
010	IIOF0	IIOF0	
011	IIOF1	IIOF1	
100	IIOF2	IIOF2	From external pins 110F0-110F3
101	IIOF3	IIOF3	
110	TIMO	TIMO	From timero TIMO and TIMI
111	TIM1	TIM1	From uners time and timit

Table 11–11. DMA Channels 2 to 5 (DMA2 to DMA5) Unified-Mode Synchronization Interrupts

[†] The *x* in DMA*x* represents the DMA channel number and also the number for the corresponding ICRDY *x* and OCRDY*x* interrupts. For example, an 001₂ in both DMA2 READ and DMA5 WRITE would enable interrupts ICRDY2 and OCRDY5, respectively. All other viable bit values (010₂ to 111₂) are the same (as shown in the table) for DMA2 through DMA5.
[‡] DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

Figure 11–26. DIE Register Bit Functions for DMA Split Mode

	31	30	29	28	27	26	25	24	23	22	21	20
	DMA	5 Primary	Write	DMA5	5 Auxiliary	Read	DMA	4 Primary	Write	DMA4	4 Auxiliary	Read
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	19	18	17	16	15	14	13	12	11	10	9	8
I	DMA	3 Primary	Write	DMA3	8 Auxiliary	Read	DMA	2 Primary	Write	DMA2	2 Auxiliary	Read
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
_	7		6	5		4	3		2	1		0
I	DMA	1 Primary	Write	DMA1	Auxiliary	Read	DMA	0 Primary	Write	DMA	0 Auxiliary	/ Read
	R/W		R/W	R/W		R/W	R/W		R/W	R/W		R/W

R = Read W = Write

Table 11–12. DMA Channels 0 and 1 (DMA0 and DMA1) Split-Mode Synchronization Interrupts

Bit Value	Interi	rupt Enabled	l at DMA0 oi	r DMA1	
(in DMA0 or DMA1)	DMA0 Auxiliary Read	DMA0 Primary Write	DMA1 Auxiliary Read	DMA1 Primary Write	Interrupt Source for DMA Synchronization
0 0†	None	None	None	None	
01	ICRDY0	OCRDY0	ICRDY1	OCRDY1	From communication port
10	IIOF0	IIOF1	IIOF2	IIOF3	From external pins IIOF0-IIOF3
11	TIM0	TIM0	TIMO	TIM0	From timer TIM0

[†] DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

Table 11–13. DMA Channels 2 to 5 (DMA2 to DMA5) Split-Mode Synchronization Interrupts

	Interrupt Enable	ed at DMA2–DMA5†	
Bit Value (in DMA2 to DMA5)	DMA <i>x</i> Auxiliary Read [†]	[†] DMA <i>x</i> Primary Write [†]	Interrupt Source for DMA Synchronization
000‡	None	None	
001	ICRDY <i>x</i> †	OCRDY <i>x</i> †	From communication port
010	IIOF0	IIOF0	
011	IIOF1	IIOF1	
100	IIOF2	IIOF2	From external pins IIOF0-IIOF3
101	IIOF3	IIOF3	
110	TIMO	TIMO	From timero TIMO and TIMI
111	TIM1	TIM1	From uniers time and time

[†] The x in DMAx represents the DMA channel number and also the number for the corresponding ICRDY x and OCRDY x interrupts. For example, an 001₂ in both DMA2 READ and DMA5 WRITE would enable interrupts ICRDY2 and OCRDY5, respective-

ly. All other viable bit values (010₂ to 111₂) are the same (as shown in the table) for DMA2 through DMA5.

[‡] DMA channel halts (no read or write operation proceeds) if DMA synchronous transfer is used.

11.10.2 Synchronization Mode Bits

Table 11–3 and Table 11–4 describe how the bit values of the SYNC MODE field of the DMA channel control register determine synchronization in unified and split mode, respectively:

- \Box No synchronization (SYNC MODE = 00₂)
- Source synchronization
 - for unified mode (SYNC MODE = 01₂)
 - for split mode (SYNC MODE = 10_2)

Destination synchronization

- for unified mode (SYNC MODE = 10₂)
- for split mode (SYNC MODE = 01_2)
- \Box Source and destination synchronization (SYNC MODE = 11₂)

When the 'C4x DMA is in split mode, the primary channel supports write (or destination) synchronization transfers only, and the auxiliary channel supports read (or source) synchronization transfers only. In split mode, bits 6 and 7 of the DMA channel control register (as shown in Table 11–3) are used to control channel synchronization:

- Bit 6 controls primary write channel synchronization (destination synchronization).
- Bit 7 controls auxiliary read channel synchronization (source synchronization).

DMA transfer rate in synchronization mode is explained in subsection 11.11.2, *DMA Transfer Rate in Synchronization Mode*, on page 11-55.

No Synchronization

When SYNC MODE = 00_2 , no synchronization is performed. The DMA performs reads and writes whenever it has the priority to use the DMA bus. All interrupts are ignored. Note the difference between this mode and having the zero value in the DIE read or write fields. Having zeros in the DIE register read/ write fields results in a total DMA halt if synchronization is used, whereas SYNC MODE = 00_2 leaves the DMA channel running freely. Figure 11–27 shows the mechanism used when SYNC MODE = 00_2 .

Figure 11–27.No DMA Synchronization



Source Synchronization

When SYNC MODE = 01_2 (for unified mode) or when SYNC MODE = 10_2 (for auxiliary channel in split mode), the DMA coprocessor is synchronized to the source (see Figure 11–28). A read will not be performed until an interrupt is received by the DMA channel. Then, all DMA interrupts are disabled globally. However, no bits in the DMA interrupt enable register are changed.

Figure 11–28. DMA Source Synchronization



Destination Synchronization

When SYNC MODE = 10_2 (for unified mode) or when SYNC MODE = 01_2 (for primary channel in split mode), the DMA channel is synchronized to the destination. A write is not performed until an interrupt is received by the DMA channel. Figure 11–29 shows the synchronization mechanism.

In unified mode, the read is performed without waiting for the interrupt. However, in split mode, the read occurs only when the interrupt enabling the write is received. This avoids a lock situation that could happen if the primary channel reads but never writes out of the temporary register, because it does not receive the write interrupt. In this case, the auxiliary channel could not proceed, because the DMA internal temporary register is busy.

Figure 11–29. DMA Destination Synchronization



Source and Destination Synchronization

When SYNC MODE = 11_2 , a read is performed when a read interrupt is received, and a write is performed on the write interrupt. If a write interrupt is received before a read interrupt, the write interrupt is latched, and the DMA data write is not executed until the read is completed. Unified mode source and destination synchronization (SYNC MODE = 11_2) is shown in Figure 11-30.

If DMA split mode is selected, it reacts as two independent synchronizations for the primary (write synchronization) and auxiliary (read synchronization) channels. Figure 11–28b and Figure 11–29b show this.

When the same interrupt is selected for read and write synchronization (in either split or unified mode), one single interrupt will enable both read and write operations. Figure 11–30. Unified Mode DMA Source and Destination Synchronization



11.11 DMA Memory Transfer Timing

The 'C4x provides six DMA channels (twelve DMA channels if they are all in split mode) with a fixed/rotating priority arbitration scheme and configurable CPU/DMA priority scheme (for detailed information, see Section 11.6 for DMA internal priority schemes and Section 11.7, *CPU and DMA Coprocessor Arbitration*, for CPU and DMA priority arbitration).

The maximum data transfer rate that the 'C4x DMA sustains is one word every two cycles. The six DMA channels transfer data in a sequential time-slice fashion, rather than simultaneously, because they share common buses.

DMA memory transfer timing can be very complicated, especially if bus resource conflicts occur. However, some rules help you calculate the transfer timing for certain DMA setups. For simplification, the following subsection focuses on a single-channel DMA memory transfer timing with no conflict with the CPU or other DMA channels. You can obtain the actual DMA transfer timing by combining the calculations for single-channel DMA transfer timing with those for bus resource conflict situations.

11.11.1 Single DMA Memory Transfer Timing

When the DMA memory transfer has no conflict with the CPU or any other DMA channels, the number of cycles of a DMA transfer depends on whether the source and destination location are designated as on-chip memory, peripheral, or external ports. When the external port is used, the DMA transfer speed is affected by two factors: the external bus wait state and the read/write conflict (for example, if a write is followed by a read, the read takes two cycles). Figure 11–31 through Figure 11–33 show the number of cycles a DMA transfer requires from different sources to different destinations. Entries in the table represents the number of cycles required to do the *T* transfers, assuming that there are no pipeline conflicts. A timing diagram for the DMA transfers accompanies each figure.

DMA Memory Transfer Timing

Cycles	Т	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Source on-chip	a	R		R		R		R		R		R		R		R		R	
Destination on-chip	Ĵ		W		W		W		W		W		W		W		W		W
Source local bus		R	R	R		R	R	R		R	R	R		R	R	R			
	4		C	r			C	r			C	r		-	C	Cr			
Destination on-chip					W				W				W				W		
Source global bus		R	R	R		R	R	R		R	R	R		R	R	R			
	4		C	r			C	r			C	r			C	Cr			
Destination on-chip					W				W				W				W		

Figure 11–31. Timing and Number of Cycles for DMA	Transfers to On-Chip Destination
---	----------------------------------

Source	Destination: On-chip
On-chip	(1+1)T
Local bus	[(1+Cr)+1]T
Global bus	[(1+Cr)+1]T

Legend: T = Number of transfers Cr = Source-read wait states R = Single-cycle reads W = Single-cycle writes R R = Multicycle reads

Cycles	Т	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Source on-chip		R		R				R				R							
Destination local bus	4		W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	
					С	w			С	W			С	W			С	w	
Source local bus		R	R	R					R	R	R	R							
	2		C	r							C	Cr							
Destination local bus	-				W	W	W	W					W	W	W	W			
					-		С	W							С	W			
Source global bus		R	R	R		R	R	R		R	R	R							
	3		C	Cr			C	Cr			C	Cr							
Destination local bus	ľ				W	W	W	W	W	W	W	W	W	W	W	W			
							С	w			С	w			С	W			

Figure 11–32. Timing and Number of Cycles for DMA Transfers to a Local-Bus Destination

Source	Destination: Local Bus
On-chip	1+(2+Cw)T
Local bus	[(2+Cr)+(2+Cw)]T-1
Global bus	[(1+Cr)+(2+Cw)]+[2+max(Cr,Cw)](T-1)

Legend: T = Number of transfers Cr = Source-read wait states Cw = Destination-write wait states R = Single-cycle reads R R = Multicycle reads W W = Multicycle writes

DMA Memory Transfer Timing

Cycles	Т	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Source on-chip		R		R				R				R							
Destination global bus	4		W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	
					С	w			С	W			С	w			С	w	
Source local bus		R	R	R		R	R	R		R	R	R							
	3		C	r			C	Cr			C	r							
Destination global bus	Ŭ				W	W	W	W	W	W	W	W	W	W	W	W			
							C	W			С	W			С	W			
Source global bus		R	R	R					R	R	R	R							
	2		C	r							C	r							
Destination global bus	-				W	W	W	W					W	W	W	W			
							С	W							С	W			

Figure	11–33. Timing	and Number of	^c Cycles for DMA	Transfers to a	Global-Bus Destination
--------	---------------	---------------	-----------------------------	----------------	------------------------

Source	Destination: Global Bus
On-chip	1+(2+Cw)T
Local bus	[(1+Cr)+(2+Cw)]+[2+max(Cr,Cw)](T-1)
Global bus	[(2+Cr)+(2+Cw)]T-1

Legend:

= Number of transfers Т

Ċr = Source-read wait states = Destination-write wait states

Cw

- R = Single-cycle reads R R = Multicycle reads W W = Multicycle writes

Externally, on the global and local buses, writes take at least two cycles. However, internally, the CPU/DMA requires one cycle to perform the write to external memory. Therefore, the DMA/CPU can transfer data on the next cycle if it is not to the same external bus. For example, the DMA transfers 1024 words from internal memory RAM block 1 to a 1-wait-state memory on the global bus while the CPU runs from memory on the local bus and fetches operands from RAM block 0. The DMA transfer time is calculated from Figure 11-32 as 1 + (2+1)1024 = 1 + 3072 = 3073 cycles.

11.11.2 DMA Transfer Rate in Synchronization Mode

The synchronization mode used for transfers also affects the DMA data transfer rate. The DMA data transfer rate is slower if synchronization is used because it takes two cycles to reset the request from the interrupt. However, these two extra cycles can be absorbed if multiple DMAs are running at the same time.

In unified mode, the maximum transfer rate is one word every three cycles, using synchronization. Figure 11–34 shows the number of cycles a DMA transfer requires under unified mode with different types of synchronization. For simplification, a single-channel DMA memory transfer timing with no conflict with CPU or other DMA channels, no memory wait states, and interrupts always active, is considered.

Cycles	Т	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
No synchronization	a	R		R		R		R		R		R		R		R		R	
No synchronization	J		W		W		W		W		W		W		W		W		W
		R			R			R			R			R			R		
Read synchronization	6		F	(r		F	{r		R	(r		F	۲		F	۲		F	(r
			W		-	W			W		-	W		-	W			W	
		R		R			R			R			R						
Write synchronization	5		W			W			W			W			W				
				٧	Vr		N	/r		V	/r		V	Vr		V	Vr		
		R			R			R			R			R					
Read and write	5		F	٢		F	۲		R	٢		Rr			F	۲			
synchronization	ľ		W			W			W			W			W				
					Nr		V		Wr		/r	W		Vr		V	Vr		

Figure 11–34. Unified-Mode DMA Timing for Different Synchronizations

Synchronization	Timing
No synchronization	2T
Read synchronization	ЗТ
Write synchronization	1 + 3T
Read and write synchronization	1 + 3T

Legend:

т = Number of transfers

R = Single-cycle reads

Ŵ = Single-cycle writes

= Read flag-reset (2 cycles) = Write-flag reset (2 cycles) Rr

Wr

In split mode, the maximum transfer rate for either the primary or auxiliary channel is one word every four cycles, using synchronization. When auxiliary and primary channels are running at the same time, the two-cycle overhead for interrupt reset is absorbed, and the maximum transfer rate can be one word every two cycles. Figure 11-35 shows the number of cycles a DMA transfer requires in split mode with different types of synchronization. For simplification, a single-channel DMA memory transfer timing with no conflict with CPU or other DMA channels, no wait states, and interrupts always active, is considered.

Cycles	Т	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		R				R				R				R					
No synchronization	ß		W				W				W				W				
running)				R'				R'				R'				R'			
					W'				W'				W'		_		W'		
Primary channel		R				R				R				R					
synchronization (auxiliary channel	4		W				W				W				W				
not running)				F	° r			P	'n			F	°r			F	° r		
Auxiliary channel		R'				R'				R'				R'					
synchronization (primary channel	4		W'				W				W'				W'				
not running)				Ar				Ar				Ar					۸r		
		R				R				R				R					
Primary and auxiliary			W				W				W				W				
channel	8			F	° r			P	'n			F	P r			F	° r		
(both channels	ľ			R'				R'				R'				R'			
running)					W'				W'				W'				W'		
						A	r			A	۸r			A	۸r			A	r

Figure 11–35. Split-Mode DMA Timing for Different Synchronizations

Synchronization	Timing
No synchronization (both channels running)	2T
Primary channel synchronization (auxiliary channel not running)	4T
Auxiliary channel synchronization (primary channel not running)	4T
Primary and auxiliary channel synchronization (both channels running)	2T + 2

Legend:

Number of transfers =

= Single-cycle reads primary channel

T R R' = Single-cycle reads auxiliary channel

W

 Single-cycle writes primary channel
 Single-cycle writes auxiliary channel
 Primary channel flag reset (2 cycles) Ŵ'

Pr Ar

= Auxiliary channel flag reset (2 cycles)

11-56

Chapter 12

Communication Ports

The 'C4x offers six ('C40) or four ('C44) on-chip communication ports for interfacing with other 'C4xs and peripherals. One important feature of the ports is that they can work with the DMA coprocessor to transfer data without CPU intervention, allowing the CPU to perform other tasks.

This chapter describes the key features, memory map and registers, and operations of the communication ports of the 'C4x digital signal processor.

Topic P	'age
12.1 Features	12-2
12.2 Operational Overview	12-3
12.3 Memory Map and Registers	12-7
12.4 Port Arbitration Units (PAUs) 1	2-11
12.5 Halting of Input and Output FIFOs12	2-14
12.6 Coordinating Communication Ports With the CPU and DMA Coprocessor	2-17
12.7 Token Transfer Operation12	2-19
12.8 Word Transfer Operation12	2-22
12.9 Synchronizers 12	2-26
12.10 Module Reset 12	2-29
12.11 Tips for Using Communication Ports12	2-32

12.1 Features

Each 'C4x communication port has several key features:

- 160-MB per second bidirectional peak data transfer rates (at 40-ns cycle time)
- Simple processor-to-processor communication via eight data lines and four control lines
- □ FIFO buffering of all data transfers
- Automatic arbitration and handshaking to ensure communication synchronization
- Synchronization between the CPU or direct-memory access (DMA) coprocessor and the six communication ports via internal interrupts and internal ready signals
- Support of a wide variety of multiprocessor architectures, including rings, trees, hypercubes, bidirectional pipelines, two-dimensional Euclidean grids, hexagonal grids, and three-dimensional grids
- \Box Communication-port software reset ('C40 revisions \geq 5.0 and 'C44 only)

12.2 Operational Overview

The 'C4x contains six ('C40) or four ('C44) identical high-speed communication ports, each of which provides a bidirectional communication interface to one other 'C4x or external peripheral. Figure 12–1 shows the internal architecture of a single communication port. Each port contains the following components:

- □ Input FIFO channel provides an 8-level, 32-bit wide first-in-first-out (FIFO) input buffer that isolates the 'C4x from the port communication data bus and buffers data received from an external device via the bus.
- Output FIFO channel provides an 8-level, 32-bit wide FIFO output buffer that isolates the 'C4x from the port communication data bus and buffers data to be sent to an external device via the bus.
- Port arbitration unit (PAU) handles the arbitration tasks associated with the movement of data between a 'C4x and an external device via the port communication data bus. The PAU is described in detail in Section 12.4, *Port Arbitration Units (PAUs)*, on page 12-11.
- Communication port control register (CPCR) allows you to control the communication port functions and data transfer operations between a 'C4x and an external device via the communication port data bus.
- Communication-port software reset register ('C44 and 'C40 rev ≥ 5.0) allows you to flush the input FIFO and output FIFO levels of a communication port. This is explained in subsection 12.3.4, *Communication Port Soft*ware Reset Register, on page 12-10.

A communication port transmits each of the 32-bit words stored in its output FIFO on a byte-to-byte basis. Because the control and data lines are bidirectional, each 'C4x must have ownership of the communication port data bus before starting a word transfer. A simulated *token* is used to designate bus ownership: the communication port that has the token owns the communication port data bus and can transmit data.




12-4



Figure 12–2. 'C4x Communication-Port Interface-Connection Example

Figure 12–2 is an example of two 'C4x DSPs connected via their communication ports. This simple communication interface consists of the following bidirectional control and data lines:

- CREQx communication-port token request. A 'C4x activates this signal to request the use of the communication-port data bus.
- CACKx communication-port token acknowledge. A 'C4x activates this signal to relinquish ownership of the communication-port data bus upon receiving a CREQx from another 'C4x.
- CSTRBx communication-port strobe. A sending 'C4x activates this signal to indicate that it has placed a valid data byte on the communication port data bus.
- CRDYx communication-port ready. A receiving 'C4x activates this signal to indicate that it has received a data byte via the communication port data bus.
- CxD(7–0) communication-port data bus. This bus carries data bidirectionally, one byte at a time, between two 'C4xs or between a 'C4x and some other device.

12.2.1 Token Transfer Operation

To transfer a token, the PAUs in the two 'C4xs cooperate to generate the signals and control sequences necessary to ensure orderly data transfers at the highest possible rate. To avoid conflicts on the bus, the PAUs arbitrate bus ownership, allowing only one DSP to transmit at any given time. The PAU that owns the token can relinquish bus ownership when the other 'C4x has data to send. The signals \overline{CREQx} and \overline{CACKx} handle the handshaking arbitration between the two DSPs in two steps:

- 1) The PAU that does not own the data bus (CxD(7–0)) activates CREQx to request bus ownership.
- 2) The PAU owning the bus then activates CACKx to acknowledge the request and relinquish bus ownership to the requesting PAU.

In this manner, these signals transfer a token (or priority) from one PAU to another, and the PAU receiving the token gains ownership of the bus. See Section 12.7, *Token Transfer Operation*, for a detailed description of token transfer.

12.2.2 Data Transfer Operation

A data transfer operation takes four basic steps to complete:

- The CPU or DMA coprocessor of the sending DSP writes a 32-bit data word to the output FIFO (of a communication port) via a memory-mapped address (listed in Figure 12–3).
- 2) The communication port then places the 32-bit data word on CxD(7–0) on a byte-to-byte basis (LS byte first), activating CSTRBx to signal the receiving communication port that the bus contains a valid data byte.
- Upon receiving each data byte, the receiving communication port activates CRDYx to indicate that it has received the data byte.
- After receiving the 4 bytes of a 32-bit word, the CPU or DMA coprocessor of the receiving DSP can then read the data from the input FIFO via a memory-mapped address (listed in Figure 12–3).

Each of the input and output FIFOs can buffer a maximum of eight 32-bit words.

Buffering provided by the input and output FIFOs is essential. This buffering allows for a high degree of decoupling of computation and communication overhead. When 'C4xs A and B are connected via their communication ports, the effective length of the FIFOs becomes 16 levels. This occurs because the output path from A to B is the concatenation of the eight levels of the output FIFO of A with the eight levels of the input FIFO of B. This also applies for the output path from B to A.

12.3 Memory Map and Registers

Figure 12–3 shows the memory map for the 'C4x communication-port control registers (CPCRs) and their associated input FIFOs and output FIFOs. The lowest three addresses of each port's 16-address block are mapped to a corresponding CPCR, and its associated input and output FIFOs. Fields (bits) within a CPCR are shown in Figure 12–4.

Figure 12–3. Communication-Port Memory Map



[†] This feature is only available on the 'C44 and on the 'C40 (revision 5.0 and above).

Communication Ports 12-7

12.3.1 Communication-Port Control Register (CPCR)

Figure 12-4 shows the format of a 'C4x CPCR, which contains control and status bits for its associated communication port. The text following the figure lists the CPCR bits and fields and describes their functions.

Figure 12–4. Communication-Port Control Register (CPCR)

			10
XX	x xx	x xx xx	хх

12	11	10	9	8	7	6	5	4	3	2	1	0
IN	PUT L	EVEL		0	UTPU	IT LE\	/EL	осн	ICH	Port Dir	xx	xx
R	R	R	R	R	R	R	R	R/W	R/W	R		

Notes: 1) xx = reserved bit (read/write as zero). 2) R = read, W = write.

Reserved	Undefined				
PORT DIR	Port Direction . This bit determines the direction of data transfer operations for the communication port.				
	PORT DIR = 0: port is in the output mode.				
	PORT DIR = 1: port is in the input mode.				
	This is a read-only bit. It is not possible to change the port direction under software control.				
ICH	Input Channel Halt. Write a 1 to ICH to halt the input channel.				
	Clear ICH to 0 when the input channel is to be unhalted.				
	The input channel cannot signal externally when it is ready to receive.				
осн	Output Channel Halt . Write a 1 to this bit to halt the output channel immediately.				
	However, the communication port is still able to accept a token request from the input channel.				
	Clear this bit to 0 to allow the output channel to transfer data.				

OUTPUT LEVEL	Output FIFO Level . Contents of this 4-bit field: 0000 ₂ (0): indicates an empty output FIFO.
	0001 ₂ (1) through 0111 ₂ (7): indicates the number of full positions in the output FIFO.
	1111 ₂ (15): indicates a full output FIFO.
	An empty output buffer (OUTPUT LEVEL = 0000_2) sends an unlatched, positive level-triggered interrupt (OCEMPTY = 1) to the CPU. When the CPU or DMA coprocessor writes to the empty output FIFO, OCEMPTY is cleared to 0 and remains in that state until the buffer is again empty. An output FIFO with one or more empty levels also sends an unlatched, positive level-trig- gered interrupt (OCRDY = 1) to the CPU and the DMA coprocessor. This condition causes a READY/NOT READY signal to be generated when the CPU or DMA coprocessor attempts to write to the output FIFO. See Section 12.6, <i>Coordinating Communication Ports With the CPU and DMA Coproces-</i> <i>sor</i> , on page 12-17, for details.
INPUT LEVEL	Input FIFO level . Contents of this 4-bit field: 0000_2 (0): indicates an empty input FIFO.
INPUT LEVEL	Input FIFO level . Contents of this 4-bit field: 0000_2 (0): indicates an empty input FIFO. 0001_2 (1) through 0111_2 (7): indicates the number of full positions in the input FIFO.
INPUT LEVEL	Input FIFO level . Contents of this 4-bit field: 0000_2 (0): indicates an empty input FIFO. 0001_2 (1) through 0111_2 (7): indicates the number of full positions in the input FIFO. 1111_2 (15): indicates a full input FIFO.
INPUT LEVEL	Input FIFO level. Contents of this 4-bit field: 0000_2 (0): indicates an empty input FIFO. 0001_2 (1) through 0111_2 (7): indicates the number of full positions in the input FIFO. 1111_2 (15): indicates a full input FIFO. A full input FIFO (INPUT LEVEL = 1111_2) sends an unlatched, positive level- triggered interrupt (ICFULL = 1) to the CPU. When the CPU or DMA coprocessor reads from the full input FIFO, ICFULL is cleared to 0 and re- mains in that state until the FIFO is again full. An input FIFO with one or more full levels also an unlatched, positive level-triggered interrupt (ICRDY = 1) to the CPU and the DMA coprocessor. This condition causes a READY/NOT READY signal to be generated when the CPU or DMA coprocessor attempts to read from the input FIFO.

12.3.2 Input-Port Register

This read-only register contains the contents of position 0, the oldest value of the input FIFO. If this register is written to, its contents remain unchanged. Reading from an empty input FIFO causes the CPU or DMA operation to stall and to halt the peripheral bus.

12.3.3 Output-Port Register

This write-only register interfaces to position 7 (the newest value) of the output FIFO. If this register is read, its contents remain unchanged, and the value read is undefined.

If an output FIFO that is full is written to, the peripheral-bus interface latches the word, and returns a *not ready* signal. This condition disappears when an empty position appears in the output FIFO and the data on the bus is transferred to the FIFO.

12.3.4 Communication-Port Software Reset Register

The input and output FIFO levels for a communication port can be flushed by writing at least two back-to-back values to its communication-port-software reset address as specified in Table 12–1. The communication port reset feature does not affect the status of the external pins.

Table 12–1. Communication-Port Software Reset Address ('C44 and 'C40 \geq 5.0)

COMMUNICATION PORT	SOFTWARE RESET ADDRESS
0†	0x0100043
1	0x0100053
2	0x0100063
3†	0x0100073
4	0x0100083
5	0x0100093

[†] These ports are available only in the 'C40.

Example 12–1 shows a method for resetting a communication port.

Example 12–2. Communication Port Reset

;; RESET1:Flu	ushes FIFOs da	ata	for communication port 1;
; RESET1 push	AR0	;	Save registers
push	RO	;	
push	RC	;	
ldhi	010h , AR0	;	Set AR0 to base address of COM 1
or	050h , AR0	;	
FLUSH:rpts	1	;F	lush FIFO data with back-to-back write
sti	R0,*+AR0(3)	;	
rpts	10	;	Wait
nop		;	
ldi	*+ARO(0),RO	;	Check for new data from other port
and	01FE0h , R0	;	
bnz	FLUSH	;	
pop	RC	;	Restore registers
pop	R0	;	
pop	AR0	;	
rets		;	Return

12.4 Port Arbitration Units (PAUs)

The PAU arbitrates between two devices to determine which device has possession of the communication port data bus at any given time. This arbitration uses CREQ and CACK signals to pass the bus ownership token back and forth between two devices connected via their communication ports. Token transfer operation is covered in detail in Section 12.7, *Token Transfer Operation*.

After system reset, half of the communication channels associated with a particular 'C4x have token ownership (communication ports 0, 1, 2), and the other half (communication ports 3, 4, 5) do not.

The PAU is a synchronous state machine with four states, as shown in Table 12–2. These states are not software-accessible by the CPU or the DMA coprocessor.

PAU State	Summary	PAU Status
State 0: Idle with token	1.PAU has token (PORT DIR = 0). 2.Channel not in use.	The PAU currently has possession of the bus own- ership token, and its associated communication channel is not in use. Under this condition, the PORT DIR bit of the associated CPCR is 0 (output). This is the state of communication ports 0, 1, and 2 after system reset.
State 1: Idle without token	 1.PAU does not have token (PORT DIR = 1). 2.Token not requested by PAU (OUTPUT LEVEL = 0). 	The PAU currently does not have possession of the bus ownership token and has not requested the token. Under this condition, the PORT DIR bit equals 1 (input), and the OUTPUT LEVEL field equals 0 (empty output FIFO). This is the state of communication ports 3, 4, and 5 after system re- set.
State 2: Active	 1.PAU has token (PORT DIR = 0). 2.Channel is in use (OUTPUT LEVEL ≠ 0). 	The PAU currently has possession of the bus own- ership token, and its associated communication channel is in use. Under this condition, the PORT DIR bit equals 0 (output), and the OUTPUT LEVEL field <i>does not</i> equal 0).
State 3: Waiting for token	1.PAU does not have token (PORT DIR = 1). 2.Token requested by PAU (OUTPUT LEVEL \neq 0).	The PAU currently does not have the bus owner- ship token but has requested it. Under this condi- tion, the PORT DIR bit equals 1 (input), and the OUTPUT LEVEL field <i>does not</i> equal 0.

Table 12-2. PAU State Definitions

Figure 12–5 shows the state diagram and controlling equations for state transitions.

To place data on the communication port data bus, the PAU must arbitrate between two types of requests:

- On-chip requests to output data in the output FIFO (shown as BUSRQ = 1 in Figure 12–5), and
- External requests received via the CREQ line (shown as TOKRQ = 1 in Figure 12–5).

Figure 12–5. Communication-Port Arbitration-Unit State Diagram



To further examine the port arbitration scheme represented in Figure 12–5, consider a data transfer operation from 'C4x A to 'C4x B. The transfer begins with PAU A in state 0 (idle with token) and PAU B in state 1 (idle without token). If PAU A receives a request (BUSRQ = 1) from its output buffer to use the communication-port data bus, it allows the output buffer to transmit one word immediately and enter state 2 (active). After the output buffer transmits one word, it removes the bus request (BUSRQ = 0), and PAU A returns to state 0 (idle with token).

If PAU B receives a request from its output buffer to use the bus, it activates \overline{CREQ} to request the token from PAU A. PAU A detects this request via the state variable TOKRQ=1 and then activates the \overline{CACK} line to transfer the bus ownership token to PAU B. PAU B then generates an internal bus acknowledge (BUSAK = 1) to indicate that it has gained bus ownership. As a result of this token transfer operation, PAU A enters state 1 (idle without token), and PAU B starts the word transfer and enters state 2 (active).

To prevent any communication port from monopolizing the communicationport bus, the PAU always returns to state 0 (idle with token) and checks for a token request (\overline{CREQ} active) from the external device after each word transfer. If the token request is active, the token is passed to the requesting device so that it can transmit a word. As long as 'C4x A and 'C4x B have information to send in their output FIFOs, they alternate use of the data bus to provide a bidirectional data path.

If a token request is received at the end of a word transfer and the sender 'C4x has another word in the output FIFO to send, two situations can occur:

- □ If the CREQ going low signal is received before CRDY low is received for the last byte, the sender 'C4x releases the token at the end of the current word transfer.
- □ If the CREQ going low signal is received after or at the same time as CRDY goes low from the last byte, the sender 'C4x continues owning the token; only after transferring the next word, will it release token ownership.

In summary, token transfer occurs only on word boundaries. The 'C4x will not release the token until the transfer of the four bytes completes.

12.5 Halting of Input and Output FIFOs

The 'C4x can halt the input FIFO, or the output FIFO, or both at word boundaries.

To halt an input FIFO, write a 1 to bit 3 (ICH) of the communication port control register (CPCR). This bit also can be read to determine if the port is halted or is able to receive. Write a 0 to the ICH bit to unhalt the input FIFO.

To halt an output FIFO, write a 1 to bit 4 (OCH) of the communication port control register (CPCR). This bit also can be read to determine whether the port is halted or is able to transmit. Write a 0 to the OCH bit to unhalt the output FIFO. The halt/unhalt operations are discussed in the following subsections. A summary is provided in Table 12–3.

Halted/Unhalted	If the Port Has Token	If the Port Does Not Have Token		
Input halted Output unhalted	a. Will not release token	a. If the halt signal is present when the input FIFO finishes receiving a word, the port will not signal ready when the first byte of a new word is received (transfer frozen). If the halt signal is received with no word reception in progress, the port receives one word and then halts.		
	b. Will transmit data	b. If halted after the first byte is received, the port receives the rest of the word and then halts the input.		
Input unhalted	a. Will not transmit data	a. Will receive data		
Output halted	b. If halted after the first byte is sent, it completes the word transfer and then halts the output.	b. Will not request token		
	c. Will release token			
Input halted Output halted	a. Will not release token	a. If the halt signal is present when the input FIFO finishes receiving a word, the port will not signal ready when the first byte of a new word is received (transfer frozen). If the halt signal is received with no word reception in progress, the port receives one word and then halts.		
	b. Will not transmit data	b. If halted after the first byte is received, if the port receives the rest of the word and then halts the input.		
	c. If halted after the first byte is sent, it completes the word transfer and then halts the output	c. Will not request token		

Table 12–3. Summary of Input and Output FIFO Halting

12.5.1 Input FIFO Halt Operation

The goal of input FIFO halting is to halt the input FIFO as soon as possible, without losing the data being input.

A communication port with an input FIFO that is either halted or is full does not respond to $\overrightarrow{\text{CSTRB}}$ low with $\overrightarrow{\text{CRDY}}$ low or acknowledge a token request with $\overrightarrow{\text{CACK}}$ low when $\overrightarrow{\text{CREQ}}$ low is received. This assures that the communication port's output channel remains open.

The communication-port logic checks whether an input FIFO halt signal has been written to the CPCR register only after finishing receiving a word. This implies:

- If the communication port receives an input halt signal when there is no word reception in progress, the input FIFO does not halt immediately; it waits to receive one word and then halts. This is the case of an input FIFO halt after reset.
- If the halt signal is written to the CPCR register while a word is being received, the input FIFO receives the rest of the current word and then halts the input. At this point, the data transfer is frozen until the input FIFO is unhalted or a system reset occurs. If the input FIFO is unhalted later, the transfer continues without any loss of data.

Notice that even when an input FIFO is halted, you can still read the words previously stored in the input FIFO.

If a communication port's input FIFO is halted during a token request from the communication port to which it is connected, then the token request is acknowledged before the input FIFO halts.

12.5.2 Output FIFO Halt Operation

Output FIFO halting is analogous to input FIFO halting and occurs also at word boundaries. Assume that 'C4x A's output FIFO has OCH = 1. Then the output FIFO will be halted on the basis of its current state.

If communication port A does not have the token:

- The output FIFO is halted immediately, and no request is made for the token.
- If the communication port requesting the token is halted after sending the CREQ signal low, the communication port still accepts the token and halts immediately after that.

If communication port A has the token:

- □ If it is currently transmitting a word, then after the current word is transmitted, the output FIFO is halted and no new transfers occur.
- □ If it is not currently transmitting a word, then the output FIFO halts immediately and no transfer occurs.
- If the input FIFO is not halted and the output FIFO is halted, then communication port A transfers the token when requested by communication port B.
- If the input FIFO is halted and the output FIFO is halted, then communication port A does not transfer the token when requested by communication port B.

If the communication port still has the token when it comes out of the halted state, it can transmit data if necessary. If it needs the token, it will arbitrate for the token as usual.

In summary, a halted output FIFO does not transmit but releases the token if the input FIFO is not halted.

12.6 Coordinating Communication Ports With the CPU and DMA Coprocessor

The communication ports support synchronization with two types of signals:

- A ready/not ready signal that can halt CPU and DMA accesses to a communication port
- Interrupts that can be used to signal the CPU and DMA

The simplest form of synchronization is based on a ready/not-ready signal. If the DMA or CPU attempt to read an empty input FIFO or write to a full output FIFO, a not-ready signal is returned, and the DMA or CPU continues to read or write (halting the peripheral bus) until a ready signal is received. The ready signal for the output channel is OCRDY (output channel ready), which is also an interrupt signal. The ready signal for the input channel is ICRDY (input channel ready), which is also an interrupt signal.

In the interrupt form of synchronization, each communication port generates four different interrupt signals, as listed below (interrupt vector locations for these are shown in Figure 7–2):

- □ ICFULL (input channel full): indicates that the input FIFO has eight words.
- ICRDY (input channel ready): indicates that at least one word is in the input FIFO.
- OCRDY (output channel ready): indicates that at least one word space is available in the output FIFO.
- OCEMPTY (output channel empty): Indicates that the output FIFO is empty.

The CPU can respond to all four of these interrupt signals. The DMA coprocessor can respond only to the ICRDY and OCRDY interrupt signals. Each DMA channel can respond only to the ICRDY and OCRDY signals coming from its own communication port; that is, DMA channel *i* can synchronize only with ICRDY*i* and OCRDY*i*.

Notice that none of the four communication-port interrupt signals has flags in the IIF register. These four communication-port status signals (ICFULL, ICRDY, OCRDY, and OCEMPTY) can be obtained by checking the input and output levels in the communication port control register (CPCR) with logical instructions. For example, to poll for an ICFULL condition, bit 12 can be tested for a bit value equal to 1. See subsection 12.3.1, *Communication-Port Control Register (CPCR)*, on page 12-8, for more information about checking for communication-port conditions.

Maximum Communication Port Sustained Transfer Rate. The maximum data transfer rate of any single communication port in a 50-MHz 'C4x is 20 M Bytes/s. This rate can be easily achieved under CPU or DMA coprocessor control, as long as data is sent to the output FIFO at least at this rate. However, when multiple communication ports are transmitting simultaneously, this may not be the case. For example, the DMA memory-to-memory maximum transfer rate is 50 M bytes/s (one read-write sequence every two cycles). The DMA can handle up to two communication ports transmitting at their full speed. For more than two communication ports, the DMA becomes the bottleneck, regardless of how many DMA channels are used. The CPU can perform two reads and two writes in two cycles by using parallel instructions, achieving a 100-M Bytes/s transfer rate. For more than five communication ports, the CPU becomes the bottleneck.

12.7 Token Transfer Operation

Token transfer operation requires handshaking of signals through pins CREQ and \overline{CACK} . This is illustrated in Figure 12–6. For clarity, a suffix identifies the signals at each processor end. For example, \overline{CREQ} b denotes the \overline{CREQ} signal at the processor B end. Table 12–4 lists the handshaking events. Steps in the table are shown by numbers in Figure 12–6.

Notice that an overlap feature is built into CREQ, CSTRB, and CRDY when a token is transferred between two 'C4x communication ports. This overlap will cause these signals to drive high (at both ends), ensuring that neither end is susceptible to floating or low-noise signals. For example, in Figure 14–23, CSTRB is an output before CREQ goes high, and in Figure 14–24, CSTRB becomes an input only after CREQ goes high. Both 'C4xs drive communication port lines for a period of 0.5 H1/H3, but this is not a problem, because they are both driving high; as a result, there is no current from one device to the other.

For this reason, the clocks of two 'C4xs connected together must be within a 2:1 ratio (at most, one 'C4x can be twice as fast as the other). If this guideline is not followed, the overlap will last too long, and the 'C4x with the faster clock may start driving low before the current bus master has relinquished that line. This will cause signal contention that could damage communication port drivers.

There is no limit on the time period between CREQ and CACK. The 'C4x can perform token transfer with a slow non-'C4x device, as long as correct hand-shaking of CREQ and CACK is maintained and there is no signal contention.

To avoid bus contention problems, you should understand which event triggers the switch of the direction (input-to-output or output-to-input) of each of the communication port bidirectional lines. This is especially important when you attempt to build a communication port interface to a non-'C4x device or when you work with very long 'C4x links. For example, the data lines and CSTRB should not be driven after CACK goes low. If they are, this could cause a bus conflict.

An implementation of a hardware token forcer can be found in the *Commu*nication Ports chapter of the TMS320C4x General-Purpose Applications User's Guide.



Figure 12–6. Token Transfer Operation

Note: For an explanation of Type 1 delay, see Section 12.9, Synchronizers.

Table 12–4. Token Transfer Sequence

Event No.†	Description
0	Initially, A has the token and is idle.
1	B wants to send data and requests the token by bringing CREQb low.
2	After a transmission line time delay, A sees the token request when CREQa goes low.
3	After a type 1 delay from CREQa falling, A releases token ownership and acknowledges the re- quest by bringing CACKa low.
4	After a transmission line time delay, B sees the acknowledgement from A when \overline{CACK} b goes low.
5	A switches CRDYa from high impedance to high after CACKa falling.
6	A puts CDa(7–0) in high impedance after CACKa falling.
7	B switches CSTRBb from high impedance to high after CACKb falling.
8	B brings CREQb high after a type 1 delay from CACKb falling.
9	After a transmission line time delay, A sees CREQa go high.
10	A switches \overline{CREQ} a from high impedance to high after receiving a high on \overline{CREQ} a.
11	A brings CACKa high after CREQa goes high.
12	A puts \overline{CACK} a in high impedance after \overline{CREQ} a goes high and after \overline{CACK} a goes high.
13	A puts CSTRBa in high impedance after CREQa goes high.
14	B puts CREQb in high impedance after CREQb goes high.
15	B switches CACKb from high impedance to high after CREQb goes high.
16	B puts CRDYb in high impedance after CREQb goes high.
17	B switches CDb from input to output after CREQb goes high and starts driving an undefined value.
18	B drives the first byte onto CDb(7–0) on H1 rising (plus analog delay) after CREQb goes high.
19	B brings $\overline{\text{CSTRB}}$ blow on the second H1 rising (plus analog delay) after $\overline{\text{CREQ}}$ brising.
20	After a transmission time delay, A sees the first byte on CDa(7–0).
21	After a transmission time delay, A sees $\overline{\text{CSTRB}}$ a go low, signaling valid data.
22	A reads the data and then brings CRDYa low.

† Event numbers correspond to numbers in Figure 12–6.

12.8 Word Transfer Operation

The C4x communication ports transfer words on a byte-to-byte basis (LS byte is transmitted first). Byte transfer operation requires handshaking of signals through pins CSTRB and CRDY. This is illustrated in Figure 12–7. For clarity, a suffix identifies the signals at each processor end. For example, CSTRB denotes the CSTRB signal at the processor B end. Table 12–5 lists the handshaking events. Steps in the table are shown by numbers in Figure 12–7.

Byte transmission is totally asynchronous, and the communication-port transfer rate can be higher than one byte per cycle. The exception is for the first byte. Notice that on the first byte, the data lines are set up in relation to an H1 synchronization (output FIFO advance). The first byte appears on a different H1 edge, depending on the transmit mode used. If the communication port is in continuous transmit mode (no token exchanged), the first data byte appears synchronous to the H1 falling edge before \overline{CSTRB} going low. That is, the data appears one half of one H1 cycle before \overline{CSTRB} falls. If a token transfer occurs, the first byte appears synchronous to the rising edge of H1 before \overline{CSTRB} going low. That is, data appears one H1 cycle before \overline{CSTRB} falls.

Subsequent bytes and $\overline{\text{CSTRB}}$ high become valid from the falling edge of $\overline{\text{CRDY}}$. Because both of these signals are caused by the same event but have different internal paths, their delay values are not exactly the same but are very close.

During back-to-back write cycles, a type 2 synchronizer is used between CRDY low to CSTRB low before byte 0 (first byte) of the next word is transmitted. Communication port synchronizers are explained in Section 12.9, *Synchronizers*, on page 12-26.

Even if the availability of data is granted, do not tie CSTRB or CRDY to ground. For each byte transfer, there must be a CSTRB and CRDY handshake. The 'C4x must see the transitions in the CSTRB and CRDY signals to advance its internal byte counter.

If an input buffer becomes full, it will not activate CRDY at the beginning of the transmission of the first byte that would overflow the buffer. This condition prevents data transfer operations until the situation is resolved. When the receiver reads the full input buffer, CRDY falls, and the next FIFO position is made available.

Notice in Figure 12–7 that after CRDYb goes low (byte 3 has been received), B drives an undefined value temporarily on CDb (7–0) (event 12 in Figure 12–7) before driving byte 0 of the new word.

Figure 12–7. Word Transfer Operation



Note: B0' = byte 0 of a new word.

Table 12–5. Word Transfer Sequence

Event No.†	Description
0	B owns the token and has data to transmit.
1	B drives the first byte onto CDb(7–0) on H1 falling (plus analog delay)‡.
2	B brings CSTRBb low on H1 rising (plus analog delay)§.
3	After a transmission line time delay, A sees $\overline{\text{CSTRB}}$ a go low, signaling valid data.
4	A reads the data and then brings CRDYa low
5	After a transmission line time delay B sees $\overline{CRDY}b$ go low, signaling data has been read.
6	B brings CSTRBb high after CRDYb goes low.
7	B drives the next byte on CDb(7–0) after \overline{CRDY} b goes low.
8	After a transmission line time delay, A sees CSTRBa go high.
9	A brings CRDYa high after CSTRBa goes high.
10	After a transmission line time delay, B sees CRDYb go high.
11	B brings CSTRBb low after CRDYb goes high.
	Events 3 through 11 repeat twice for bytes 2 and 3 (asynchronous handshaking)
12	B drives an undefined byte on CDb(7–0) after CRDY goes low.

[†] Event numbers correspond to numbers in Figure 12–7.

‡ If this is the first word the token is received, this transition occurs after CREQb goes high (See event 18 in Table 12–4).

\$ If this is the first word after the token is received, this transition occurs on the second H1 rising after CREQb goes high (See event 19 in Table 12–4).

CSTRB Width Restrictions

In 'C4x device revisions lower than 3.0, the width of the CSTRB low pulse between word boundaries should not exceed 1.0 H1/H3 at the receiving 'C4x end. If it does, the receiver 'C4x byte counter that has looped back to byte 0 between word boundaries will see this low and recognize CSTRB as the next valid byte, effectively slipping a byte. This is not a problem unless you are working with very long distances or with external devices. If you are, use flipflops to locally shorten the CSTRB at the receiver end while returning a valid CRDY width to the sender. Wide widths at the sender are not a problem. Chapter 7, *Interfacing Communication Ports*, the *TMS320C4x General-Purpose Applications User's Guide* shows a circuit to shorten the CSTRB low pulse. In 'C4x device revisions 3.0 or higher, no CSTRB width restriction exists.

Note:

See Chapter 7, Interfacing Communication Ports, in the TMS320C4x General-Purpose Applications User's Guide for a detailed description of the word transfer operation when interfacing a 'C4x communication port with a non-'C4x device.

12.9 Synchronizers

H1/H3 synchronization is required during word transfer boundaries and during token transfers. Three types of synchronizers are used in the port arbitration unit:

□ Type-one synchronizers cause delays that vary from 1 to 2 machine clock from the receiving of an input on a pin until the response on output pin (ignoring analog delays). An input is recognized when H1 is high; then it is passed through an H3-high/H1-high series of delays. The response occurs at the start of the following time H3 is high.

The minimum type-one synchronizer delay of 1 machine clock will occur when the input changes just before H1 goes low. This delay is shown in Figure 12–8.

The maximum type-one synchronizer delay of 2 machine clocks will occur when the input changes just after H1 goes low. This delay is shown in Figure 12–9.

Figure 12–8. Type-One Synchronizer Minimum Delay



Figure 12–9. Type-One Synchronizer Maximum Delay



□ Type-two synchronizers cause delays that vary from 1.5 to 2.5 machine clock from the receiving of an input on a pin until the response on an output pin (ignoring analog delays). An input is recognized when H1 is high; then it is passed through an H3-high/H1-high/H3-high series of delays. The response occurs at the start of the following time H1 is high.

The minimum type-two synchronizer delay of 1.5 machine clocks occurs when the input changes just before H1 goes low. This delay is shown in Figure 12–10.

The maximum type-two synchronizer delay of 2.5 machine clocks occurs when the input changes just after H1 goes low. This delay is shown in Figure 12–11.

Figure 12–10. Type-Two Synchronizer Minimum Delay



Figure 12–11. Type-Two Synchronizer Maximum Delay



□ Type-three synchronizers cause delays that vary from 0.5 to 1.5 machine clocks from the receiving of an input on a pin until the response on output pin (ignoring analog delays). An input is recognized when H1 is high; then it is passed through an *H3-high* delay. The response occurs at the following time H1 is high.

The minimum type-three synchronizer delay of 0.5 machine clock cycles will occur when the input changes just before H1 goes low. This delay is shown in Figure 12–12.

The maximum type-three synchronizer delay of 1.5 machine clocks will occur when the input changes just after H1 goes low. This delay is shown in Figure 12–13.

Figure 12–12. Type-Three Synchronizer Minimum Delay



Figure 12–13. Type-Three Synchronizer Maximum Delay



Table 12–6 shows the types of synchronizer delays for communication port signals.

Table 12–6. Communication-Port Signals and Synchronizer Delays

Input Signal to Output Signal	Delay Type	Min. Delay (clock cycles)	Max. Delay (clock cycles)
$\overline{CREQ}\downarrow$ to $\overline{CACK}\downarrow$	One	1	2
CACK↓ to CREQ↑	One	1	2
$\overline{\text{CRDY}}{\downarrow}$ to CD valid between back-to-back word transfers	One	1	2
$\overline{\text{CRDY}}{\downarrow}$ to $\overline{\text{CSTRB}}{\downarrow}$ between back-to-back word transfers	Two	1.5	2.5
$\overrightarrow{CACK}\downarrow$ to \overrightarrow{CSTRB} switch from input to an output high.	Three	0.5	1.5

12.10 Module Reset

This section explains the status of the 'C4x communication ports after powerup and during and after system reset.

The recommended reset sequence in a multiprocessing system is described in Chapter 1, *Processor Initialization and Program Control*, in the *TMS320C4x General-Purpose Applications User's Guide*.

After powerup, the status depends on the RESET pin:

If RESET is low, the 'C4x is in reset immediately, and the description under "at reset" (below) applies.

If RESET is not low, the 'C4x device is in an unknown stage. The communication port signals can be in a combination of states.

At reset (while RESET = 0), the communication port pins are all put in the high-impedance state. The input and output channels both assume an empty state, causing all values in the input and output buffers to be lost. Pullup resistors should be used on all control lines to ensure that they are logic high if reset is not applied at the same time in interconnected 'C4xs.

After reset (after the rising edge of RESET), communication ports 0,1, and 2 are configured as output ports and assume the following states:

- The PAU is reset to state 0: The PAU has the bus ownership token and is idle.
- □ The pin status (see Figure 12–14) is set as follows:
 - The CxD(7–0) signals start driving an undefined value.
 - The CACK and CSTRB signals go to 1 (inactive). CREQ and CRDY continue to be high-impedance.

Note:

The individual communication port software reset feature only flushes the FI-FOs, but does not have any effect on the communication port external pins.

The communication port control register gets a 0h value:

- PORT DIR = 0: the communication port is configured for an output operation.
- INPUT LEVEL = 0: The input FIFO is empty.
- OUTPUT LEVEL = 0: The output FIFO is empty.
- ICH = 0: The input FIFO is not in its halted state.
- OCH = 0: The output FIFO is not in its halted state.

□ ICRDY = 0: The input FIFO is empty and is not ready to be read from.

OCRDY = 0: The input FIFO is not full and is ready to be written to.





After reset (after rising edge of **RESET**), communication ports 3,4, and 5 are configured as input ports and assume the following states:

- PAU is reset to state 1: The PAU does not have the bus ownership token, and the token is not requested.
- ☐ The pin status (see Figure 12–15) is set as follows:
 - \blacksquare CxD(7–0) continue to be high-impedance.
 - CREQ and CRDY signals go to 1 (inactive). CACK and CSTRB continue in high impedance.
- The communication port control register gets a value of 04h.

PORT DIR = 1: the communication port is configured for an input operation:

- INPUT LEVEL = 0: The input FIFO is empty.
- OUTPUT LEVEL = 0: The output FIFO is empty.
- ICH = 0: The input FIFO is not in its halted state.
- OCH = 0: The output FIFO is not in its halted state.
- □ ICRDY = 0: The input FIFO is empty and is not ready to be read from.
- OCRDY = 0: The input FIFO is not full and is ready to be written to.

Figure 12–15. Post-Reset State for an Input Port



At reset, ports 0, 1, and 2 are configured as output ports (PORT DIR = 0), and ports 3, 4, and 5 are configured as input ports (PORT DIR = 1). When you interconnect the ports of two 'C4x devices, connect the port of one 'C4x to a port of the other 'C4x that would be in the *opposite* direction at reset in other words, connect any one of port 0, 1, or 2 connected to any one of port 3, 4, or 5.

If your system configuration requires connection of input-to-input communication ports or output-to-output communication ports, refer to Chapter 7, *Interfacing Communication Ports*, in the *TMS320C4x General-Purpose Applications User's Guide* for an implementation of a token forcer.

12.11 Tips for Using Communication Ports

When you design systems that use the communication ports, there are considerations to keep in mind:

- ☐ At reset, ports 0–2 are set for transmit and ports 3–5 are set for receive. When connecting communication ports between 'C4x devices, make sure you connect transmit ports to receive ports and receive ports to transmit ports. Otherwise, unpredictable results may occur.
- Signal quality is very important. Make sure you design your board to minimize noise from other components. See the section entitled Signal Considerations in the communications port chapter of the TMS320C4x General-Purpose Applications User's Guide for more information.
- Do not read from an empty input FIFO. This will cause the CPU or DMA operation to stall and to halt the peripheral bus.
- Do not write to an unconnected communication port. If a port's transmit FIFO is full and the port can't transmit, an additional write to the port's FIFO will halt the peripheral bus.
- The clocks of two 'C4xs connected together must be within a 2:1 ratio (at most, one 'C4x can be twice as fast as the other). If this guideline is not followed, the 'C4x with the faster clock may start driving low before the current bus master has relinquished that line. This will cause signal contention that could damage communication port drivers. This restriction does not apply when connecting to a non-'C4x device.
- When you design an interface to a non-'C4x device, the non-'C4x device should mimic the asynchronous handshaking operation of a 'C4x communication port. See Word Transfer Considerations in the TMS320C4x General-Purpose Applications User's Guide for more information on interfacing to non-'C4x devices.

Note:

See Section 7.4, *Design Tips*, in the *TMS320C4x General-Purpose Applications User's Guide* for more tips for using communication ports.

Chapter 13

Timers

The 'C4x has two general-purpose timer modules that time events, generate pulses, and interrupt the CPU or DMA coprocessor.

This chapter provides you with information about:

- ☐ The components of the timers
- ☐ The control registers of the timers
- The operation of the timers
- □ The interrupts generated by the timers

Topic

Page

13.1	Overview of the Timers : 13-2
13.2	Timer Pins :
13.3	Timer Control Registers :
13.4	Timer Pulse Generation : 13-9
13.5	Timer Interrupts : 13-11
13.6	Selecting CLKSRC and FUNC Values :
13.7	Using TCLKx as General-Purpose I/O Pins :
13.8	Configuring a Timer :

13.1 Overview of the Timers

The 'C4x has two 32-bit general-purpose timer modules. Each timer has two signaling modes and can be clocked by an internal or an external source. The timer modules can be used to send periodic signals to the 'C4x or to devices in the external world; or they can be used to count external events. Each timer has an I/O pin (TCLK) that functions as an input clock, as an output clock, or as a general-purpose I/O pin.

With an internal clock, for example, the timer can signal an external A/D converter to start a conversion, or it can interrupt the 'C4x DMA controller to begin a data transfer.

With an external clock, for example, the timer can count external events and interrupt the CPU after a specified number of events.

Each timer consists of a 32-bit counter, a comparator, an input clock selector, a pulse generator, and supporting hardware.

A timer in the 'C4x counts the cycles of a timer input clock. When that count (counter register) equals the value stored in the timer period register, it rolls over the counter to zero and produces a transition in the timer output signal.

The timer input clock can be either the H1/2 internal clock frequency of the 'C4x or an external clock on the TCLKx pin. This is determined by the CLKSRC bit in the timer control register. If an external clock is used, the timer can counter either 0-to-1 or 1-to-0 transitions depending on the value of the INV bit.

The timer output signal depends on the signalling mode selected by the C/P bit (clock or pulse mode). See Section 13.4, *Timer Pulse Generation*, on page 13-9, for more information about this bit.

The timer output can be routed to the TCLKx pin that can also be used as a general-purpose I/O pin.

Figure 13–1 shows the block diagram of a 'C4x timer module.

Figure 13–1. Timer Block Diagram



 † If CLKSRC =1 and FUNC = 1, this signal goes into the TCLK pin. ‡ Selector controlled by the CLKSRC bit. $^{\$}$ Maximum frequency = f(H1) / 2.6

13.2 Timer Pins

Each timer has one pin associated with the timer clock signal (TCLK) pin.

□ **TCLK.** This pin is used as a general-purpose I/O signal, as a timer output, or as an input for an external clock for a timer. Each timer has a TCLK pin: TCLK0 is connected to timer 0 and TCLK1 is connected to timer 1.

13.3 Timer Control Registers

The timers are controlled through three registers, as shown in Figure 13–2, that are mapped into the peripheral address space:

- □ Control register. This register determines the operating mode of the timer, monitors the timer status, and controls the function of the I/O pin (TCLK) of the timer.
- Period register. This register contains the number of timer input clock cycles to count. This number controls the timer output signal frequency.
- □ **Counter register**. Contains the current value of the incrementing counter. The 32-bit counter counts timer input clock cycles.

Figure 13–2.	Memor	y-Mapped	Timer I	Locations
--------------	-------	----------	---------	-----------

Register	Periphe	Peripheral Address			
	Timer 0	Timer 1			
Timer Control	100020h	100030h			
Reserved	100021h	100031h			
Reserved	100022h	100032h			
Reserved	100023h	100033h			
Timer Counter	100024h	100034h			
Reserved	100025h	100035h			
Reserved	100026h	100036h			
Reserved	100027h	100037h			
Timer Period	100028h	100038h			
Reserved	100029h	100039h			
Reserved	10002Ah	10003Ah			
Reserved	10002Bh	10003Bh			
Reserved	10002Ch	10003Ch			
Reserved	10002Dh	10003Dh			
Reserved	10002Eh	10003Eh			
Reserved	10002Fh	10003Fh			

13.3.1 Timer Control Register

The timer control register is located at 100020h for timer 0 and at 100030h for timer 1.

The 32-bit timer global control register contains two sets of bits:

- ☐ The timer global control bits (bits 11−6) control timer mode and monitor timer status (TSTAT).
- ☐ The TCLK pin control bits (bits 3–0) control the function of the TCLK pin, which can be used as a timer pin or as a general-purpose I/O pin.

Figure 13–3 shows the 32-bit timer global control register. Note that at reset all bits are set to 0, except for DATIN, which is set to the value read on TCLK.

Figure 13–3. Timer Control Register

32	12	11	10	9	8	7	6	5	4	3	2	1	0
	xx	TSTAT	INV	CLKSRC	C/P	HLD	GO	×	x	DATIN	DATOUT	I/O	FUNC
Note:	R=Read,	R/W , W=Write	R/W	R/W	R/W	R	R			R	R	R	R
	FL	JNC	Fur = 0, TCI	TCLK is K is conf	. The F configi igured	UNC b ured as as a tin	it cont a gei ner pi	trols neral n.	the -pui	function pose di	of the TC gital I/O p	LK pin. I in. If FU	f FUNC NC = 1,
	I/C)	Inp inpu	ut/output ut pin. If I/(t bit. If D = 0 a	I/O = 1 nd FUN	and I IC = 0	UN , the	C = n TC	0, then CLK is co	TCLK is c onfigured a	onfigure as an ou	d as an tput pin.
	DA	ATOUT	Dat	a output FOUT car	bit. DA also b	ATOUT e used	drives as ar	s TCI n inp	LK v ut to	vhen the the tim	e 'C4x is ir er.	n I/O por	t mode.
	DA	ATIN	Data input bit. Reads data from TCLK or DATOUT. A write to this bit has effect.								t has no		
	G	D	GO not edg Wha furtl	bit. Rese held, the e of the tin en GO = 0 her define	ets and counte mer inp 0, the ti es this t	starts t er is zer out clock mer is pit.	he tim oed a k. The not af	ner c nd b GO fecte	ouni egir bit is ed. S	ter. Whens increr ters cleared Section 1	en GO = 1 menting or d on the sa 13.8, <i>Conf</i>	and the n the ne ame risir <i>iguring i</i>	timer is xt rising Ig edge. A <i>Timer,</i>
	н	HLD Counter hold bit. When HLD = 0, the counter is disabled and held in its content state. If the timer is driving TCLK, the state of TCLK is also held. The internal divide-by-two counter is also held so that the counter can contine where it left off when HLD is set to 1. The timer registers can be read a modified while the timer is being held. RESET has priority over HLD. Section 13.8, <i>Configuring A Timer,</i> shows the effect of writing to GO and HLD, a shows the result of a write using specified values of the GO and HLD bits the timer global control register.								its cur- ∋ld. The continue ∋ad and Section LD, and D bits in			

GO (Bit 6)	HLD (Bit 7)	Result
0	0	All timer operations are held. No reset is performed.
0	1	Timer proceeds from state before write.
1	0	All timer operations are held, including zeroing of the counter. The GO bit is not cleared until the timer is taken out of hold.
1	1	Timer resets and starts.

- **C**/ \overline{P} **Clock/pulse mode control.** When C/ \overline{P} = 1, clock mode is chosen, and the signaling of the TSTAT status bit and TCLK pin will have a 50 percent duty cycle. When C/ \overline{P} = 0, the TSTAT status bit and TCLK pin will be active for one H1 cycle during each timer period (see Figure 13–4).
- CLKSRC Timer input clock source select bit. Specifies the source of the timer input clock. When CLKSRC = 1, an internal clock with frequency equal to one-half the H1 frequency is used as the timer input clock, and the INV bit has no effect. When CLKSRC = 0, an external signal from the TCLK pin is used as the timer input clock. The external clock is synchronized internally, thus allowing external asynchronous clock sources that do not exceed the specified maximum allowable external clock frequency of f(H1)/2.6.
- INV Inverter control bit. If an external clock is used as the timer input clock and INV= 1, the external clock is inverted as it goes into the counter. If the output of the pulse generator (TSTAT) is routed to TCLK and INV = 1, the output is inverted before it goes to TCLK. If INV = 0, no inversion is performed on the input or output of the timer. The INV bit has no effect, regardless of its value, when TCLK is used in I/O port mode.
- **TSTAT Timer status bit.** This bit tracks the output of the timer and sets a CPU interrupt on a transition from 0 to 1. A write has no effect.

13.3.2 Timer Period Register

The timer period register is located at 100028h for timer 0 and at 100038h for timer 1.

The 32-bit timer period register contains the number of timer input clock cycles to count. This number controls the frequency of the timer output signal.
The frequency of timer signaling is determined by the frequency of the timer input clock and the period register. The following equations are valid with either an internal or an external timer clock:

 $f(pulse mode) = f(timer clock) \div period register$ $f(clock mode) = f(timer clock) \div (2 \times period register)$

This register is cleared to 0 at reset.

13.3.3 Timer Counter Register

The timer period register is located at 100024h for timer 0 and at 100034h for timer 1.

The 32-bit timer counter register increments with each cycle of the timer input clock. The timer counter can be incremented on the rising edge (INV = 0) or on the falling edge (INV = 1) of an externally generated timer input clock (CLKSRC = 0). With an internally generated timer input clock (CLKSRC = 1), the timer counter increments on the rising edge only. The timer counter is zeroed whenever its value equals that of the period register.

This register is cleared to 0 at reset.

13.3.4 Boundary Conditions in the Control Registers

Certain boundary conditions, such as a zero in the period register and an overflow of the counter, affect timer operation. These conditions are listed as follows:

- □ When the period and counter registers are zero, the operation of the timer depends on the C/ \overline{P} mode selected. In pulse mode (C/ \overline{P} = 0), TSTAT is set and remains set. In clock mode (C/ \overline{P} = 1), the width of a cycle is 2/f(H1), and external clocks are ignored.
- When the counter register is not 0 and the period register = 0, the counter will count until it reaches its maximum 32-bit value (0FFFF FFFFh), roll over to 0, and then function as described in the preceding bullet.
- When the counter register is set to a value greater than the value of the period register, the counter reaches its maximum 32-bit value (0FFFF FFFFh), rolls over to 0, and continues.

Note:

Writes from the peripheral bus override register updates from the counter and new status updates to the control register.

13.4 Timer Pulse Generation

The timer pulse generator (see Figure 13–1) can generate several different TSTAT signals. These signals can be inverted (set by the INV bit) into the timer output signal. The two basic pulse generation modes are pulse mode and clock mode, as shown in Figure 13–4. You can select the mode with the C/\overline{P} bit of the timer global control register. In both modes, an internal clock source has a frequency of $f(H1) \div 2$, and an external clock source has a maximum frequency of $f(H1) \div 2.6$. In pulse mode $(C/\overline{P} = 0)$, the width of the pulse is 1/f(H1). In clock mode $(C/\overline{P}=1)$, the width of the pulse is the period register divided by the frequency of the timer input clock.





Note: TINT is the timer interrupt signal generated whenever TSTAT transitions from 0 to 1.

Timers 13-9

The rate of the timer output (TSTAT) is determined by the frequency of the timer input clock and the period register. The following equations are valid with either an internal or an external timer clock:

In pulse mode: f(TSTAT) = f(timer input clock) ÷ period register

In clock mode: $f(TSTAT) = f(timer input clock) \div (2 \times period register)$

If the period register equals zero, refer to subsection 13.3.2, *Timer Period Register*.

Figure 13–5 provides some examples of TSTAT and timer output (INV = 0) when the period register is set to various values and clock or pulse mode is selected. Timer input clock is generated internally ($f(H1 \div 2)$).

Figure 13–5. Timer Output Generation Examples



13.5 Timer Interrupts

Each timer can send an interrupt to the CPU when the TSTAT signal transitions from 0 to 1. Timer 0 sends TINT0 and timer 1 sends TINT1.

TINTO. This interrupt uses the interrupt vector at IVTP + 002h. It has a priority level of two, which is second only to NMI and RESET.

TINT1. This interrupt uses the interrupt vector at IVTP + 02Bh. It has a the lowest priority level of all interrupts.

13.5.1 Timer Interrupts and Their Vectors

TINT0 corresponds to timer 0. This interrupt uses the interrupt vector at IVTP + 002h. It has a priority level of two, which is second only to NMI and RESET.

TINT1 corresponds to timer 1. This interrupt uses the interrupt vector at IVTP + 02Bh. It has a the lowest priority level of all interrupts.

13.5.2 Timer Interrupt Operation

A timer interrupt is generated whenever TSTAT transitions from a zero to a one. The frequency of timer interrupts depends on whether the timer is set up in pulse mode or clock mode.

In pulse mode, the interrupt frequency is:

f(interrupt)=f(input timer clock) ÷ period register

In clock mode, the interrupt frequency is:

 $f(interrupt) = f(input timer clock) \div (2 \times period register)$

If the period register equals zero, see subsection 13.3.4, *Boundary Conditions in the Control Registers*, on page 13-8, for more information.

The timer interrupt can be used to interrupt either the CPU or the DMA coprocessor.

The timer interrupt enable bits for the CPU are found in the IIE register. Bit 0 in the IIE corresponds to TINT0, and bit 1 corresponds to TINT1. For more information about the IIE register, see subsection 3.1.9, *CPU Internal Interrupt Enable Register (IIE)*, on page 3-11.

The timer interrupt enable bits for the DMA control register are found in the DIE register. Several bits in this register control how each DMA channel responds

Timers 13-11

to the timers. For more information about the DIE register, see subsection 13.3.4, *Boundary Conditions in the Control Registers*.

13.5.3 Considerations When Using a Timer Interrupt

The main consideration when using a timer to interrupt the CPU is the priority needed for the operation. If the timer operation has a low priority compared to other devices, then use timer 1, since that timer's interrupt has the lowest priority of all interrupts. If, on the other hand, the timer operation has a high priority compared to other devices, then use timer 0, since that timer's interrupt is second in priority only to an NMI.

13.6 Selecting CLKSRC and FUNC Values

The timer can receive its input and send its output in several different modes, depending on the setting of CLKSRC, FUNC, and \bar{I}/O . The four timer modes of operation are defined by the values of CLKSRC and FUNC in the global control register.

13.6.1 CLKSRC = 1 and FUNC = 0.

If CLKSRC = 1 and FUNC = 0 (see Figure 13–6), the timer input comes from the internal clock. Interrupts can still be generated during the transition of TSTAT from 0 to 1. The internal clock is not affected by the INV bit in the global control register. In this mode, TCLK is connected to the I/O port control and can be used as a general-purpose I/O pin. If $\overline{I}/O = 0$, TCLK is configured as a general-purpose input pin whose state can be read in DATIN. DATOUT has no effect on TCLK or DATIN. If $\overline{I}/O = 1$, TCLK is configured as a general-purpose output pin. DATOUT is placed on TCLK and can be read in DATIN.

Figure 13–6. Timer Configuration With CLKSRC=1 and FUNC=0



13.6.2 CLKSRC=1 and FUNC=1.

If CLKSRC = 1 and FUNC = 1 (see Figure 13–7), the timer input comes from the internal clock, and the timer output goes to TCLK. You can invert the value on TCLK by setting INV to 1. Also, the value of TCLK can be read in DATIN.

Figure 13–7. Timer Configuration With CLKSRC = 1 and FUNC = 1



Timers 13-13

13.6.3 CLKSRC = 0 and FUNC = 0

If CLKSRC = 0 and FUNC = 0 (see Figure 13–8), the timer can still generate interrupt signals and is driven according to the status of the \overline{I}/O bit:

- ☐ If Ī/O = 0, the timer input comes from TCLK. You can invert the value read from TCLK by setting INV to 1, and the value of TCLK can be read through DATIN.
- If I/O = 1, TCLK is an output pin; both TCLK and the timer are driven by DATOUT. All 0-to-1 transitions of DATOUT increment the counter. INV has no effect on DATOUT. The value of DATOUT can be read through DATIN.

Figure 13–8. Timer Configuration With CLKSRC = 0 and FUNC = 0



13.6.4 CLKSRC = 0 and FUNC = 1

If CLKSRC = 0 and FUNC = 1 (see Figure 13–9), TCLK drives the timer. If INV = 0, all 0-to-1 transitions of TCLK increment the counter. If INV= 1, all 1-to-0 transitions of TCLK increment the counter. The value of TCLK can be read through DATIN.

Figure 13–9. Timer Configuration With CLKSRC = 0 and FUNC = 1



13.7 Using TCLKx as General-Purpose I/O Pins

When FUNC = 0, TCLKx can be used as an I/O pin. Figure 13–10 and Figure 13–11 show how the TCLKx is connected when it is configured as a general-purpose I/O pin. In Figure 13–10, the I/O bit equals 0 and TCLK is configured as an input pin whose value can be read in the DATIN bit. In Figure 13–11, the I/O bit equals 1 and TCLK is configured as an output pin that outputs the value you wrote in the DATOUT bit.

Figure 13–10. TCLK as an Input (I/O = 0)



Figure 13–11. TCLK as an Output (I/O = 1)



13.8 Configuring a Timer

Configuring a timer requires three basic steps:

- Halt the timer by clearing to 0 the GO and HLD bits of the timer global-control register. To do this, write a 0 to the timer global-control register. Note that the timers are halted on RESET.
- 2) Configure the timer via the timer global-control register (with GO=HLD=0), the timer counter register, and timer period register, if necessary.
- Start the timer by setting the GO and HLD bits of the timer global-control register to 1.

Example 13–1 shows how to set up the 'C4x timer to generate the maximum frequency clock through the TCLKx pin.

Example 13–1. Maximum Frequency Timer Clock Setup

```
TITLE MAXIMUM FREQUENCY TIMER CLOCK SETUP
*
  THIS EXAMPLE SHOWS HOW TO SET UP TIMER TO GENERATE MAXIMUM
*
  FREQUENCY TIMER CLOCK USING INTERNAL CLOCK. WHERE
*
   "TIMER_REGISTER" SECTION IS LOCATED FROM 100020h.
TIM0_CTL_REG .usect "TIMR_REGISTER",4
TIM0_CNT_REG .usect "TIMR_REGISTER",4
TIM0_PRD_REG .usect "TIMR_REGISTER", 8
             .text
             .
                   0,R0
             LDI
                 R0,@TIM0_PRD_REG
             STI
             LDI
                    3C1H,R0
             STI
                   R0,@TIM0_CTL_REG
             .end
```

Chapter 14

Assembly Language Instructions

The 'C4x assembly language instruction set supports numeric-intensive, signal processing, and general-purpose applications. The instructions are organized into these major groups: load-and-store, two- or three-operand arithmetic/logical, parallel, program control, and interlocked operations instructions. The addressing modes used with the instructions are described in Chapter 6.

The 'C4x instruction set can also use one of 20 condition codes with any of the 10 conditional instructions, such as LDF*cond*. This chapter defines the condition codes and flags.

The assembler allows optional syntax forms to simplify the assembly language for special-case instructions. These optional forms are listed and explained.

Each of the individual instructions is described and listed in alphabetical order. An example instruction (on pages 14-23 through 14-25) demonstrates the special format used and explains its content.

This chapter discusses these topics:

Торіс

Page

14.1	Instruction Set 14-2
14.2	Condition Codes and Flags 14-12
14.3	Individual Instruction Descriptions14-16

14.1 Instruction Set

The 'C4x instruction set is exceptionally well-suited to digital signal processing and other numeric-intensive applications. All instructions are a single machine word long, and most instructions take a single cycle to execute. In addition to multiply and accumulate instructions, the 'C4x possesses a full complement of general-purpose instructions.

The instruction set contains 145 instructions organized into the following functional groups:

- Load-and-store
- Two-operand arithmetic/logical
- Three-operand arithmetic/logical
- Program control
- Interlocked operations
- Parallel operations

Each of these groups is discussed in the succeeding subsections.

14.1.1 Load-and-Store Instructions

The 'C4x supports 24 load-and-store instructions (see Table 14–1). These instructions can:

- Load a word from memory into a register
- Store a word from a register into memory
- Manipulate data on the system stack
- Transfer data between primary register and expansion register

Two of these instructions can load data conditionally. This is useful for locating the maximum or minimum value in a data set. See Section 14.2 for detailed information on condition codes.

Instruction	Description	Instruction	Description
LBb†	Load byte (signed)	LDPK†	Load DP register immediate
LBUb [†]	Load byte (unsigned)	LHw†	Load half-word signed
LDA [†]	Load address register	LHUw [†]	Load half-word unsigned
LDE	Load floating-point exponent	LWL <i>ct</i> [†]	Load word left-shifted
LDEP†	Load integer, expansion-file register to primary register	LWR <i>ct</i> †	Load word right-shifted
LDF	Load floating-point value	POP	Pop integer from stack
LDF <i>cond</i>	Load floating-point value conditionally	POPF	Pop floating-point value from stack
LDHI†	Load 16-bit unsigned immediate into 16 MSBs	PUSH	Push integer on stack
LDI	Load integer	PUSHF	Push floating-point value on stack
LDI <i>cond</i>	Load integer conditionally	STF	Store floating-point value
LDM	Load floating-point mantissa	STI	Store integer
LDPE [†]	Load integer, primary register to expan- sion file register	STIK†	Store integer immediate

Table 14–1. Load-and-Store Instructions

[†]The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

14.1.2 Two-Operand Instructions

The 'C4x supports a complete set of 43 two-operand arithmetic and logical instructions. The two operands are the source and destination. The source operand can be a memory word, a register, or a constant. The destination operand is always a register.

These instructions provide integer, floating-point, or logical operations, and multiprecision arithmetic. Table 14–2 lists these instructions.

Table 14–2. Two-0	Dperand I	Instructions
-------------------	-----------	--------------

Instruction	Description	Instruction	Description
ABSF	Absolute value of a floating-point number	MPYF [†]	Multiply floating-point values
ABSI	Absolute value of an integer	MPYI†	Multiply integers
ADDC [†]	Add integers with carry	MPYSHI†‡	Multiply signed integer, 32-MSB product
ADDF [†]	Add floating-point values	MPYUHI†‡	Multiply unsigned integer, 32-MSB product
ADDI [†]	Add integers	NEGB	Negate integer with borrow
AND [†]	Bitwise logical-AND	NEGF	Negate floating-point value
ANDN [†]	Bitwise logical-AND with complement	NEGI	Negate integer
ASH [†]	Arithmetic shift	NORM	Normalize floating-point value
CMPF [†]	Compare floating-point values	NOT	Bitwise logical-complement
CMPI [†]	Compare integers	OR [†]	Bitwise logical-OR
FIX	Convert floating-point value to integer	RCPF [‡]	Reciprocal floating point
FLOAT	Convert integer to floating-point value	RND	Round floating-point value
FRIEEE‡	Convert IEEE floating-point format to 2s- complement floating-point format	ROL	Rotate left
LSH [†]	Logical shift	ROLC	Rotate left through carry
MB <i>ct</i> ‡	Merge byte, left shifted	ROR	Rotate right
MH <i>ct</i> ‡	Merge half-word, left shifted	RORC	Rotate right through carry

[†]Two- and three-operand versions

[‡] The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

Instruction	Description	Instruction	Description
RSQRF‡	Reciprocal of square root, floating-point	SUBRF	Subtract reverse floating-point value
SUBB†	Subtract integers with borrow	SUBRI	Subtract reverse integer
SUBC	Subtract integers conditionally	TOIEEE‡	Convert 2s complement to IEEE format
SUBF [†]	Subtract floating-point values	TSTB [†]	Test bit fields
SUBI†	Subtract integer	XOR [†]	Bitwise exclusive-OR
SUBRB	Subtract reverse integer with borrow		

Table 14–2. Two-Operand Instructions (Continued)

[†] Two- and three-operand versions. [‡] The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

14.1.3 Three-Operand Instructions

Most instructions contain two or three operands. The 19 three-operand instructions allow the 'C4x to read two operands from memory or the CPU register file in a single cycle and store the results in a register. The following differentiates the two- and three-operand instructions:

- Two-operand instructions have one source operand (or shift count) and a destination operand.
- □ Three-operand instructions may have two source operands (or one source operand and a count operand) and a destination operand. A source operand can be a memory word, a register or a constant. The destination of a three-operand instruction is always a register.

Table 14–3 lists the instructions that have three-operand versions. Note that the 3 in the mnemonic can be omitted from three-operand instructions (see subsection 14.3.2).

Instruction	Description	Instruction	Description
ADDC3	Add with carry	MPYI3	Multiply integers
ADDF3	Add floating-point values	MPYSHI3 [†]	Multiply signed integer, 32-MSB product
ADDI3	Add integers	MPYUHI3 [†]	Multiply unsigned integer, 32-MSB product
AND3	Bitwise logical-AND	OR3	Bitwise logical-OR
ANDN3	Bitwise logical-AND with comple- ment	SUBB3	Subtract integers with borrow
ASH3	Arithmetic shift	SUBF3	Subtract floating-point values
CMPF3	Compare floating-point values	SUBI3	Subtract integers
CMPI3	Compare integers	TSTB3	Test bit fields
LSH3	Logical shift	XOR3	Bitwise exclusive-OR
MPYF3	Multiply floating-point values		

Table 14–3. Three-Operand Instructions

⁺ The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

14.1.4 Program Control Instructions

The program-control instruction group consists of all of those instructions (24) that affect program flow. The repeat mode allows repetition of a block of code (RPTB and RPTBD) or of a single line of code (RPTS). Both standard and delayed (single-cycle) branching are supported. Several of the program control instructions are capable of conditional operations (see Section 14.2 for detailed information on condition codes). Table 14–4 lists the program control instructions.

	Table	14-4.	Program	Control	Instructions
--	-------	-------	---------	---------	--------------

Instruction	Description	Instruction	Description
Bcond	Branch conditionally (standard)	LAJ†	Link and jump
B <i>cond</i> AF†	Branch conditionally delayed and annul if false	LAJ <i>cond</i> †	Link and jump conditional
B <i>cond</i> AT†	Branch conditionally delayed and annul if true	LAT <i>cond</i> †	Link and trap conditional
B <i>cond</i> D	Branch conditionally (delayed)	NOP	No operation
BR‡	Branch unconditionally (standard)	RETI <i>cond</i>	Return from interrupt conditionally
BRD‡	Branch unconditionally (delayed)	RETI- <i>cond</i> D _†	Return from trap or interrupt, delayed
CALL [‡]	Call subroutine	RETS <i>cond</i>	Return from subroutine conditionally
CALLcond	Call subroutine conditionally	RPTB [‡]	Repeat block of instructions
DBcond	Decrement and branch conditionally (standard)	RPTBD	Repeat block, delayed
DB <i>cond</i> D	Decrement and branch conditionally (delayed)	RPTS	Repeat single instruction
IACK	Interrupt acknowledge	SWI	Software interrupt
IDLE	Idle until interrupt	TRAPcond	Trap conditionally

[†] The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

 \ddagger Operand addressing mode is incompatible with 'C3x.

14.1.5 Interlocked Operations Instructions

The interlocked operations instructions support multiprocessor communication and the use of external signals to allow for powerful synchronization mechanisms. They also guarantee the integrity of the communication and result in a high-speed operation. Refer to Chapter 7 for examples of the use of interlocked instructions.

Table 14–5. Interlocked Operations Instructions

Instruction	Description	Instruction	Description
LDFI	Load floating-point value, interlocked	STFI	Store floating-point value, interlocked
LDII	Load integer, interlocked	STII	Store integer, interlocked
SIGI	Signal, interlocked		

14.1.6 Parallel Operations Instructions

The parallel-operations instructions group makes a high degree of parallelism possible. Some of the 'C4x instructions can occur in pairs that are executed in parallel. These instructions offer the following features:

- Parallel loading of registers
- Parallel store
- Parallel arithmetic operations
- Arithmetic/logical instructions used in parallel with a store instruction.

Each instruction in a pair is entered as a separate source statement. The second instruction in the pair must be preceded by two vertical bars (||). Table 14–6 lists the valid instruction pairs.

Table 14–6. Parallel Instructions

(a) Parallel Arithmetic With Store Instructi	ons
--	-----

Mnemonic	Description
ABSF STF	Absolute value of a floating-point number and store floating-point value
ABSI STI	Absolute value of an integer and store integer
ADDF3 STF	Add floating-point values and store floating-point value
ADDI3 STI	Add integers and store integer
AND3 STI	Bitwise-logical AND and store integer
ASH3 STI	Arithmetic shift and store integer
FIX STI	Convert floating-point to integer and store integer
FLOAT STF	Convert integer to floating-point value and store floating-point value
FRIEEE STF [†]	Convert IEEE floating-point format and store
LDF STF	Load floating-point value and store floating-point value
LDI STI	Load integer and store integer

[†] The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

Table 14-6. Parallel Instructions (Concluded)

(a) Parallel Arithmetic With Store Instructions (Continued)

Mnemonic	Description
LSH3 STI	Logical shift and store integer
MPYF3 STF	Multiply floating-point values and store floating-point value
MPYI3 STI	Multiply integer and store integer
NEGF STF	Negate floating-point value and store floating-point value
NEGI STI	Negate integer and store integer
NOT STI	Complement value and store integer
OR3 STI	Bitwise-logical OR value and store integer
STF STF	Store floating-point values
STI STI	Store integers
SUBF3 STF	Subtract floating-point value and store floating-point value
TOIEEE STF [†]	Convert to IEEE format and store
SUBI3 STI	Subtract integer and store integer
XOR3 STI	Bitwise-exclusive OR values and store integer

(b) Parallel Load Instructions

Mnemonic	Description
LDF LDF	Load floating-point
LDI LDI	Load integer

(c) Parallel Multiply and Add/Subtract Instructions

Mnemonic	Description
MPYF3 ADDF3	Multiply and add floating-point
MPYF3 SUBF3	Multiply and subtract floating-point
MPYI3 ADDI3	Multiply and add integer
MPYI3 SUBI3	Multiply and subtract integer

[†]The 'C4x instruction set is a superset of the 'C3x instruction set. The instructions marked are 'C4x-specific.

14.1.7 Illegal Instructions

The 'C4x has no illegal instruction detection mechanism. Fetching an illegal (undefined) code may result in the execution of an undefined operation. If TI TMS320 floating-point software tools are used, no illegal opcodes can be generated. An illegal opcode can only be generated by the misuse of the tools, by an error in the ROM code, or by a defective RAM.

14.2 Condition Codes and Flags

The 'C4x provides 20 condition codes (00000–10100, excluding 01011) that can be used with any of the conditional instructions, such as RETS*cond* or LDF*cond*. The conditions include signed and unsigned comparisons, comparisons to zero, and comparisons based on the status of individual condition flags. Note that all conditional instructions can also accept the suffix U to indicate unconditional operation.

Seven condition flags provide information about properties of the result of arithmetic and logical instructions. The condition flags are stored in the status register (ST); the effect of an instruction on a condition flag depends on the value of the SET COND field (bit 15 of the status register). The value of SET COND (0 or 1) does not affect the nature of the compare instructions (CMPF, CMPF3, CMPI3, CMPI3, TSTB, or TSTB3).

- ☐ If SET COND = 0, the ST condition flags are set if the operation's target is any extended-precision register (R0–R11).
- If SET COND = 1, the ST condition flags are **also** set if the operation's target is *any* register in the primary register file *except* the status register.

The condition flags can be modified by most instructions when either of the preceding conditions is established and either of the following two cases occurs:

- A result is generated when the specified operation is performed to infinite precision. This is appropriate for compare-and-test instructions that do not store results in a register. It is also appropriate for arithmetic instructions that produce underflow or overflow.
- ☐ The output is written to the destination register as shown in Table 14–7. This is appropriate for other instructions that modify the condition flags.

Type of Operation	Output Format
Floating-point	8-bit exponent, 1 sign bit, 31-bit fraction
Integer	32-bit integer
Logical	32-bit unsigned integer

Table 14–7. Output Value Formats

Figure 14–1 shows the condition flags in the low-order bits of the status register. Following the figure is a list of status register condition flags and descriptions on how the flags are set by most instructions. For specific details of the effect of a particular instruction on the condition flags, see the description of that instruction in subsection 14.3.3.

_	31	30	29	28	27	26	25	24	23	22	2	1 2	20	19	18	17	16
Ľ	xx	хх	хх	хх	хх	хх	xx	xx	XX	xx	x	x x	κx	xx	xx	xx	Analysis
	R	R	R	R	R	R	R	R	R	R	F	1	R	R	R	R	R
_		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Γ	SET	COND	PGIE	GIE	CC	CE	CF	PCF	RM	OVM	LUF	LV	UF	N	Z	V	С
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
٢	NOTE:	xx = re	eserved	bit.													

Figure 14–1. Status Register

R = read, W = write.

- **LUF** Latched Underflow Condition Flag. LUF is set whenever UF (floating-point underflow flag) is set. LUF can be cleared only by a processor reset or by modifying it in the status register (ST).
- LV Latched Overflow Condition Flag. LV is set whenever V (overflow condition flag) is set. Otherwise, it is unchanged. LV can be cleared only by a processor reset or by modifying it in the status register (ST).
- UF Floating-Point Underflow Condition Flag. A floating-point underflow occurs whenever the exponent of the result is less than or equal to −128. If a floating-point underflow occurs, UF is set, and the output value is set to 0. UF is cleared if a floating-point underflow does not occur.
- N Negative Condition Flag. Logical operations assign N (the state of the MSB of the output value). For integer and floating-point operations, N is set if the result is negative, and cleared otherwise. Zero is positive.
- **Z Zero Condition Flag.** For logical, integer, and floating-point operations, Z is set if the output is 0, and cleared otherwise.
- V **Overflow Condition Flag.** For integer operations, V is set if the result does not fit into the format specified for the destination (i.e., $-2^{32} \le \text{result} \le 2^{32} 1$). Otherwise, V is cleared. For floating-point operations, V is set if the exponent of the result is greater than 127; otherwise, V is cleared. Logical operations always clear V.

C Carry Flag. When an integer addition is performed, C is set if a carry occurs out of the bit corresponding to the MSB of the output. When an integer subtraction is performed, C is set if a borrow occurs into the bit corresponding to the MSB of the output. Otherwise, for integer operations, C is cleared. The carry flag is unaffected by floating-point and logical operations. For shift instructions, this flag is set to the final value shifted out; for a zero shift count, this is set to zero.

Table 14–8 lists the condition mnemonic, code, description, and flag for each of the 19 condition codes.

Table 14–8. Condition Codes and Flags

(a) Unconditional Compares

Condition	Code	Description	Flag [†]			
U 00000		Unconditional	Don't care			
(b) Unsigned C	ompares					
Condition	Code	Description	Flag [†]			
LO LS HI HS EQ NE	00001 00010 00011 00100 00101 00110	Lower than Lower than or same as Higher than Higher than or same as Equal to Not Equal to	C C OR Z ~C AND ~Z ~C Z ~Z			
(c) Signed Compares						
Condition	Code	Description	Flag [†]			
LT LE GT GE EQ NE	00111 01000 01001 01010 00101 00110	Less than Less than or equal to Greater than Greater than or equal to Equal to Not equal to	N N OR Z ~N AND ~Z ~N Z ~Z			
(d) Compare to	(d) Compare to Zero					
Condition	Code	Description	Flag [†]			
Z NZ P N NN	00101 00110 01001 00111 01010	Zero Not zero Positive Negative Nonnegative	Z ~Z ~N AND ~Z N ~N			

[†]The ~ means logical complement ("not true" condition).

Condition	Code	Description	Flag [†]
NN	01010	Nonnegative	~N
N	00111	Negative	Ν
NZ	00110	Nonzero	~Z
Z	00101	Zero	Z
NV	01100	No overflow	~V
V	01101	Overflow	V
NUF	01110	No underflow	~UF
UF	01111	Underflow	UF
NC	00100	No carry	~C
С	00001	Carry	С
NLV	10000	No latched overflow	~LV
LV	10001	Latched overflow	LV
NLUF	10010	No latched floating-point underflow	~LUF
LUF	10011	Latched floating-point underflow	LUF
ZUF	10100	Zero or floating-point underflow	Z OR UF

(e)	Compare	to	Condi	tion	FI	ags
-----	---------	----	-------	------	----	-----

 \dagger The \sim means logical complement ("not true" condition).

14.3 Individual Instruction Descriptions

This section contains the individual assembly language instructions for the 'C4x. The instructions are listed in alphabetical order. Information for each instruction includes assembler syntax, operation, operands, encoding, description, cycles, status bits, mode bit, and examples.

Definitions of the symbols and abbreviations, as well as optional syntax forms allowed by the assembler, precede the individual instruction description section. Also, an example instruction shows the special format used and explains its content.

You can find a functional grouping of the instructions, as well as a complete instruction set summary in Section 14.1. See Chapter 7, *Addressing and Stack Management*, for information on memory addressing.

14.3.1 Symbols and Abbreviations

Table 14–9 lists the symbols and abbreviations used in the individual instruction descriptions.

Table 14–9. Instruction Symbols

Symbol	Meaning
src	Source operand
src1	Source operand 1
src2	Source operand 2
src3	Source operand 3
src4	Source operand 4
dst	Destination operand
dst1	Destination operand 1
dst2	Destination operand 2
disp	Displacement
cond	Condition
count	Shift count
G	General addressing modes
T	Three-operand addressing modes
P	Parallel addressing modes
B	Conditional-branch addressing modes
ARn IRn RC RE RS ST	Auxiliary register n Index register n Extended-precision register address n Repeat count register Repeat end address register Repeat start address register Status register
C	Carry bit of status register
GIE	Global interrupt enable bit of status register
N	Trap vector
PC	Program counter
RM	Repeat mode flag
SP	System stack pointer
x	Absolute value of x
x \rightarrow y	Assign the value of x to destination y
x(man)	Mantissa field (sign + fraction) of x
x(exp)	Exponent field of x
op1 op2	Operation 1 performed in parallel with operation 2
x AND y	Bitwise-logical AND of x and y
x OR y	Bitwise-logical OR of x and y
x XOR y	Bitwise-logical XOR of x and y
~x	Bitwise-logical complement of x
x << y	Shift x to the left y bits
x >> y	Shift x to the right y bits
*++SP	Increment SP and use incremented SP as address
*SP	Use SP as address and decrement SP

Assembly Language Instructions 14-17

14.3.2 Optional Assembler Syntaxes

The assembler allows a relaxed syntax form for some instructions. These optional forms simplify the assembly language so that special-case syntax can be ignored. The following is a list of these optional syntax forms.

☐ The destination register can be omitted on unary arithmetic and logical operations when the same register is used as a source. For example,

ABSI R0, R0 *can be written as* ABSI R0 Instructions affected: ABSI, ABSF, FIX, FLOAT, NEGB, NEGF, NEGI, NORM, NOT, RND.

All three-operand instructions can be written without the 3. For example,

ADDI3 R0, R1, R2 can be written as ADDI R0, R1, R2 Instructions affected: ADDC3, ADDF3, ADDI3, AND3, ANDN3, ASH3, LSH3, MPYF3, MPYI3, OR3, SUBB3, SUBF3, SUBI3, XOR3, MPYSHI3, MPYUHI3.

This also applies to all the pertinent parallel instructions.

 All three-operand comparison instructions can be written without the 3. For example,

CMPI3 R0, *AR0 can be written as CMPI R0, *AR0

Instructions affected: CMPI3, CMPF3, TSTB3.

□ Indirect operands with an explicit 0 displacement are allowed. In three-operand or parallel instructions, operands with 0 displacement are automatically converted to no-displacement mode. For example:

LDI *+AR0(0), R1 is legal

Also

ADDI3 *+AR0(0), R1, R2 is equivalent to ADDI3 *AR0, R1, R2

Indirect operands can be written with no displacement; in which case, a displacement of 1 is assumed. For example,

LDI *AR0++(1), R0 can be written as LDI *AR0++, R0

All conditional instructions accept the suffix U to indicate unconditional operation. Also, the U can be omitted from unconditional short branch instructions. For example:

BU label can be written as B label

Labels can be written with or without a trailing colon. For example:

```
      label0: NOP

      label1
      NOP

      label2:
      (label assembles to next source line)
```

Empty expressions are not allowed for the displacement in indirect mode:				
LDI *+ARO(),RO is not legal				
Immediate-mode destination operands of BR and CALL can be written with an at (@) sign :				
BR label can be written as BR @label				
The LDP pseudo-op can be used to load a register (DP by default) with the 16 MSBs of a relocatable address as follows:				
LDP addr,REG Or LDP @addr,REG Or LDP addr				
The <i>at</i> (@) sign is optional.				
LDP generates an LDIU instruction. An immediate operand with a special relocation type is used.				
Parallel instructions can be written in either order. For example:				
ADDI				
can be written as				
can be written as				
ADDI				
The parallel bars indicating part two of a parallel instruction can be written anywhere on the line from column 0 to the mnemonic. For example:				
ADDI				
STI				
can be written as				
ADDI				
STI				
If the second operand of a parallel instruction is the same as the third (des- tination register) operand, the third operand can be omitted. This allows the writing of three-operand parallel instructions that look like normal two- operand instructions. For example,				

```
ADDI *AR0,R2,R2
|| MPYI *AR1,R0,R0
```

can be written as

ADDI *AR0,R2 || MPYI *AR1,R0

Instructions affected (applies to all parallel instructions that have a register as the second operand): ADDI, ADDF, AND, MPYI, MPYF, OR, SUBI, SUBF, XOR.

All commutative operations in parallel instructions can be written in either order. For example, the ADDI part of a parallel instruction can be written in either of two ways:

ADDI *ARO,R1,R2 Or ADDI R1,*ARO,R2

The instructions affected are parallel instructions containing any of the following: ADDI, ADDF, MPYI, MPYF, AND, OR, XOR.

Use the syntax in Table 14–10 to designate CPU registers in operands.

14.3.3 Individual Instruction Descriptions

Each assembly language instruction for the 'C4x is described in this section in alphabetical order. The description includes the assembler syntax, operation, operands, encoding, description, cycles, status bits, mode bit, and examples. Table 14–10 shows the CPU register symbols.

	Register Machine			
Register Symbol	Value (hex)	Assigned Function Name	Subsection	Page
R0	00	Extended-precision register 0	3.1.1	3-3
R1	01	Extended-precision register 1	3.1.1	3-3
R2	02	Extended-precision register 2	3.1.1	3-3
R3	03	Extended-precision register 3	3.1.1	3-3
R4	04	Extended-precision register 4	3.1.1	3-3
R5	05	Extended-precision register 5	3.1.1	3-3
R6	06	Extended-precision register 6	3.1.1	3-3
R7	07	Extended-precision register 7	3.1.1	3-3
R8	1C	Extended-precision register 8	3.1.1	3-3
R9	1D	Extended-precision register 9	3.1.1	3-3
R10	1E	Extended-precision register 10	3.1.1	3-3
R11	1F	Extended-precision register 11	3.1.1	3-3
AR0	08	Auxiliary register 0	3.1.2	3-4
AR1	09	Auxiliary register 1	3.1.2	3-4
AR2	0A	Auxiliary register 2	3.1.2	3-4
AR3	0B	Auxiliary register 3	3.1.2	3-4
AR4	0C	Auxiliary register 4	3.1.2	3-4
AR5	0D	Auxiliary register 5	3.1.2	3-4
AR6	0E	Auxiliary register 6	3.1.2	3-4
AR7	0F	Auxiliary register 7	3.1.2	3-4
DP	10	Data-page pointer	3.1.3	3-4
IR0	11	Index register 0	3.1.4	3-4
IR1	12	Index register 1	3.1.4	3-4
BK	13	Block-size register	3.1.5	3-5
SP	14	System stack pointer	3.1.6	3-5

Table 14–10. CPU Register Symbols

Assembly Language Instructions 14-21

Individual Instruction Descriptions

Register Symbol	Register Machine Value (hex)	Assigned Function Name	Subsection	Page
ST	15	Status register	3.1.7	3-5
DIE	16	DMA coprocessor interrupt enable	3.1.8	3-8
IIE	17	Internal-interrupt enable register	3.1.9	3-11
IIF	18	IIOF pins and interrupt flag register	3.1.10	3-13
RS	19	Repeat start address	3.1.11	3-16
RE	1A	Repeat end address	3.1.11	3-16
RC	1B	Repeat counter	3.1.11	3-16
IVTP	00	Interrupt-vector table pointer	3.2	3-17
TVTP	01	Trap-vector table pointer	3.2	3-17

Table 14–10. CPU Register Symbols (Continued)

Syntax	INST src, dst
	or
	INST1 src2, dst1 INST2 src3, dst2
	Each instruction begins with an assembler syntax expression. Labels may be placed either before the command (instruction mnemonic) on the same line or on the preceding line in the first column. The optional comment field that con- cludes the syntax is not included in the syntax expression. A space is required between fields (label, command, operand, and comment fields).
	The syntax examples illustrate the common one-line syntax and the two-line syntax used in parallel addressing. Note that the two vertical bars that indicate a parallel addressing pair can be placed anywhere before the mnemonic on the second line. The first instruction in the pair can have a label, but the second instruction cannot have a label.
Operands	<i>src</i> general-addressing modes (G): <i>dst</i> register (R0 – R11)
	The operands segment lists the types of operands that the instruction uses.
Opcode	

31	24 2	23	16	15	8	7	0	
000	INST	G	dst	SIC				
31	24	1 23		16	515	8 7	0	
111	NST1 INST2	dst1	000	src3	dst2	src2		

Encoding examples are shown for general addressing and parallel addressing. The instruction pair for the parallel addressing example consists of INST1 and INST2. Note that two separate opcodes are listed in this case; each instruction is 32-bits in length in the 'C4x.

Word Fields

G	<i>src</i> addressing modes
00	register (R0 – R11)
01	direct
10	indirect
11	immediate

	The wo each va sponds	ne word fields segment describes the addressing mode that correspond ach value of a word field in the opcode. The word field listed in the table co conds to the field listed under operands.						
Operation	$ src \rightarrow dst$							
	or							
	$ src2 \rightarrow dst1$ $ src3 \rightarrow dst2$							
	The instruction operation sequence describes the processing that takes place when the instruction is executed. For parallel instructions, the operation sequence is performed in parallel. Conditional effects of status register specified modes are listed for conditional instructions such as B <i>cond</i> .							
	<i>dst</i> regi	ster (any register in CPU primary-register file)						
	or							
	src2indirect (disp = 0, 1, IR0, IR1) $dst1$ register (R0–R7) $src3$ register (R0–R7) $dst2$ indirect (disp = 0, 1, IR0, IR1)							
	Operands are defined according to the addressing mode and/or the dressing used. Note that indirect addressing uses displacements a dex registers. See Chapter 6, <i>Addressing</i> , for detailed information or sing.							
Description	Instruct tents ar by the p and sup	tion execution and its effect on the rest of the processor or memory con- re described in this segment. Any constraints on the operands imposed processor or the assembler are discussed. The description parallels pplements the information given by the operation block.						
Status Bits								
	LUF	Latched Floating-Point Underflow Condition Flag. 1 if a float- ing-point underflow occurs, unchanged otherwise.						
	LV	Latched Overflow Condition Flag. 1 if an integer or floating-poin overflow occurs, unchanged otherwise.						
	UF	Floating-Point Underflow Condition Flag. 1 if a floating-point underflow occurs, 0 otherwise.						
	Ν	Negative Condition Flag. 1 if a negative result is generated, 0 otherwise. In some instructions, this flag is the MSB of the output.						
	Z	Zero Condition Flag. 1 if a zero result is generated, 0 otherwise. For logical and shift instructions, 1 if a zero output is generated, 0 otherwise.						

	V	 V Overflow Condition Flag. 1 if an integer or floating-point overflow curs, 0 otherwise. C Carry Flag. 1 if a carry or borrow occurs, 0 otherwise. For shift i tions, this flag is set to the value of the last bit shifted out; 0 fo count of 0. 					
	С						
	The se informa operati	ven condition flags are stored in ation about the properties of the rons.	the sta esult	atus register (ST). They provide or output of arithmetic or logical			
Mode Bit	OVM C OVM b	Overflow Mode Flag. In general, it value.	intege	er operations are affected by the			
Cycles	1						
	The dig	git specifies the number of cycles	s requ	ired to execute the instruction.			
Example	INST	@98AEh,R5					
		Before Instruction		After Instruction			
	DP	80h DP	80h				
	R5 (07 6690 0000h 2.30562500e + 02	R5	00 6690 0000h 1.80126593e + 00			
	Memory	v at 0080 98AEh 5CDFh 1.00001107e + 00	Men	nory at 80 98AEh 5CDFh 1.00001107e + 00			
	LUF	0	LUF	0			
	LV	0	LV	0			
	UF 🗌	0	UF	0			
	N	0	Ν	0			
	Z 🗌	0	Z	0			
	V [0	V	0			
	с 🗌	0	С	0			

The sample code presented in the above format shows the effect of the code on system pointers (e.g., DP or SP), registers (e.g., R1 or R5), memory at specific locations, and the seven status bits. The values given for the registers include the leading zeros to show the exponent in floating-point operations. Decimal conversions are provided for all register and memory locations. The seven status bits are listed in the order in which they appear in the assembler and simulator (see Section 14.2, *Condition Codes and Flags*, and Table 14–8 on page 14-14 for further information on these seven status bits).

Svntax	ABSF src. dst					
Operands	<i>src:</i> general-addressing modes <i>dst:</i> register (R0 – R11)					
Opcode						
	31 29	23 21	16	0		
	0 0 0 0 0 0 0 0	0 G	dst	STC		
Word Fields						
		G	src addres	sing modes		
		00	register (RC	9 – R11)		
		01	direct			
		10	indirect			
		11	immediate			
Operation	$ src \rightarrow dst$					
Description	The absolute value and <i>dst</i> operands a	of the <i>src</i> are assume	operand is led to be floa	oaded into the <i>dst</i> register. The <i>src</i> ting-point numbers.		
	An overflow occurs is <i>dst</i> (man) = 7FF	if <i>src</i> (man F FFFFh ar) = 8000 00 nd <i>dst</i> (exp)	00h and <i>src</i> (exp) = 7Fh. The result = 7Fh.		
Status Bits	LUFUnaffectedLV1 if a floatingUF0N0Z1 if a zero reV1 if a floatingCUnaffected	g-point ove esult is gen g-point ove	rflow occurs erated, 0 ot rflow occurs	s, unchanged otherwise herwise s, 0 otherwise		
Mode Bit	OVM operation is a	affected by	the OVM bi	t's value.		
Cycles	1					
Example	ABSF R4,R7					
·	Before I R4 05C8000F971h R7 07D251100AEh LV 0 Z 0 V 0	nstruction]-9.9033730] 5.4852725]]]	7e + 27 5e + 37	After Instruction R4 05C8000F971h -9.9033737e + 27 R7 05C7FFF068Fh 9.90337307e + 27 LV 0 Z 0 V 0		

Syntax	AB ∥ ST	BSF src2, F src3,	dst1 dst2					
Operands	src2:indirect (disp = 0, 1, IR0, IR1) $dst1:$ register (R0 - R7) $src3:$ register (R0 - R7) $dst2:$ indirect (disp = 0, 1, IR0, IR1)							
Opcode								
	31 29 1 1 0	9 0 1 0	24 23 0 <i>dst1</i>	0 0 0	16 src3	15 dst2	8 7	0 src2
Word Fields	None.							
Operation	src src	22 ightarrow dst 1 3 ightarrow dst 2	,					
Description	A floating-point absolute value and a floating-point store are performed in par- allel. All registers are read at the beginning and loaded at the end of the ex- ecute cycle. This means that if one of the parallel operations (STF) reads from a register and the operation being performed in parallel (ABSF) writes to the same register, then STF accepts as input the contents of the register before it is modified by the ABSF.							
	If <i>src2</i> and <i>dst2</i> point to the same location, <i>src2</i> is read before the write to <i>dst2</i> If <i>src3</i> and <i>dst1</i> point to the same register, <i>src3</i> is read before the write to <i>dst</i>						vrite to <i>dst2</i> . vrite to <i>dst1.</i>	
	An overflow occurs if <i>src</i> (man) = 8000 0000h and <i>src</i> (exp) = 7Fh. The result is <i>dst</i> (man) = 7FFF FFFFh and <i>dst</i> (exp) = 7Fh.							
Status Bits	 LUF Unaffected LV 1 if a floating-point overflow occurs, unchanged otherwise UF 0 N 0 Z 1 if a zero result is generated, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise C Unaffected 							
Mode Bit	OVM	operation i	s not affe	ected by C	OVM bit	value.		
Cycles	1							

Example

ABSF *++AR3(IR1),R4 STF R4,*-AR7(1)

	Before Ir	nstruction
AR3	80 9800h	
IR1	0AFh	
R4	733C0 0000h	1.79750e + 02
AR7	80 98C5h	
Data	at 80 98AFh 58B 4000h	-6.118750e + 01
Data	at 80 98C4h 0h	
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	



Syntax		erc det			
Operands	<i>src:</i> general-addressing modes <i>dst</i> : register (any register in CPU primary-register file)				
Opcode	31 0 0 0	24	23 1 G	1615 dst	8 7 0 src
Word Fields					
			G	src addressing modes	-
		•	00	register (any register in CPU primary-register file)	_
			01	direct	
			10	indirect	
			11	immediate	_
Operation	<i>src</i> -	→ dst			
Description	The a and a	bsolute value <i>st</i> operands a	of the <i>src</i> re assume	operand is loaded into the ed to be signed integers.	<i>dst</i> register. The <i>src</i>
	An ov dst=7	verflow occur 7FFFFFFh.	s if <i>src</i> = If ST(OV	8000 0000h. If ST(OVM M) = 0, the result is <i>dst</i> = 8	4) = 1, the result is 3000 0000h.
Status Bits	If ST flags a regist	(SET COND) = are modified. I ers.	= 0 and the f ST (SET	e destination register is R0 COND) = 1, they are modi) – R11, the condition fied for all destination
	LUF LV UF N Z V C	Unaffected 1 if an intege 0 0 1 if a zero re 1 if an intege Unaffected	er overflow sult is ger er overflow	v occurs, unchanged other nerated, 0 otherwise v occurs, 0 otherwise	wise
Mode Bit	ΟνΜ	operation is a	ffected by	OVM bit value.	

Cycles

1

ABSI Absolute Value of Integer

Example 1	ABSI RO,RO or ABSI RO			
	Before Instruction	-53	After Instruction R0 035h	53
Example 2	ABSI *AR1,R3			
	Before Instruction		After Instruction	
	AR1 20h		AR1 20h	
	R3 0h		R3 35h	53
	Data at 20h OFFFF FFCBh	-53	Data at 20h OFFFF FFCBh	-53

Syntax	ABSI ∥ STI	src2, dst1 src3, dst2				
Operands	src2: indirect (disp = 0, 1, IR0, IR1) dst1: register (R0 - R7) src3: register (R0 - R7) dst2: indirect (disp = 0, 1, IR0, IR1)					
Opcode						
	31	24 23 0 1 dst1	0 0 0	16 src3	15 8 dst2	7 0 src2
Word Fields	None			0.00		0.02
Operation	src2 - $ src3 \rightarrow$	→ dst1 dst2				
Description	An integer absolute value and an integer store are performed in parallel. All registers are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (ABSI) writes to the same register, then STI accepts as input the contents of the register before it is modified by the ABSI.					
	If <i>src2</i> and	<i>dst2</i> point to the	same loo	cation, <i>si</i>	rc2 is read befor	re the write to <i>dst2.</i>
	An overflo 7FFF FFF	w occurs if <i>src</i> = Fh. If ST(OVM)	= 8000 0 = 0, the i	000h. If result is	ST(OVM) = 1, <i>dst</i> = 8000 000	the result is <i>dst</i> = 0h.
Status Bits	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N 0 Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C Unaffected 					
Mode Bit	OVM operation is affected by OVM bit value.					
Cycles	1					

ABSI||STI Parallel ABSI and STI

Example

ABSI *-AR5(1),R5 || STI R1,*AR2--(IR1)

	Before Instruc	tion		After Instruction
AR5	80 99E2h		AR5 [80 99E2h
R5	0h		R5 [35h
R1	42h	66	R1 [42h
AR2	80 98FFh		AR2	80 98F0h
IR1	0Fh		IR1	0Fh
Data	at 80 99E1h 0FFFF FFCBh	-53	Data [at 80 99E1h 0FFFF FFCBh
Data	at 80 98FFh 2h	2	Data [at 80 98FFh 42h
LUF [0		LUF [0
LV [0		LV [0
UF [0		UF [0
N [0		N [0
Ζ [0		Z [0
V	0		V [0
с [0		с [0

53

66

-53

66

Syntax	ADDC src, dst			
Operands	<i>src:</i> general-addressing modes <i>dst:</i> register (any register in CPU primary-register file)			
Opcode				
	31 24	1 23	1615	8 7 0
	0 0 0 0 0 0 0 1	0 G	dst	src
Word Fields				
		G	src addressing modes	
		00	register (any register in CPU primary-register file)	
		01	direct	
		10	indirect	
		11	immediate	
Operation	$dst + src + C \rightarrow ds$	t		
Description	The sum of the <i>dst dst</i> register. The <i>ds</i>	and <i>src</i> op st and <i>src</i> o	erands and the C (carry) perands are assumed to	flag is loaded into the be signed integers.
Status Bits	If ST (SET COND) flags are modified. registers.	= 0 and the If ST (SET	e destination register is R COND) = 1, they are moc	0 – R11, the condition dified for all destination
	LUF Unaffected			

- LV 1 if an integer overflow occurs, unchanged otherwise
- **UF** 0

1

- **N** 1 if a negative result is generated, 0 otherwise
- **Z** 1 if a zero result is generated, 0 otherwise
- V 1 if an integer overflow occurs, 0 otherwise
- **C** 1 if a carry occurs, 0 otherwise

Mode Bit OVM operation is affected by OVM bit value.

Cycles

ADDC Add Integer With Carry

Example

ADDC R1,R5

	Before Instruction			After Ins	truction
R1	00FFFF 5C25h	-41 947	R1	00FFFF 5C25h	-41 947
R5	00FFFF 019Eh	-65 122	R5	00FFFE 5DC4h	-107 068
LUF	0		LUF	0	
LV	0		LV	0	
UF	0		UF	0	
Ν	0		Ν	0	
Z	0		Z	0	
V	0		V	0	
С	0		С	0	

Syntax	ADI	DC3 src2, src1, d	st		
Operands	src1 dst:	1, src2: type 1 or t register m	type 2 three-ope node (any registe	rand addressing n r in CPU primary-	nodes register file)
Opcode					
	Тур	be 1			
	31	24 23	161	5 8	7 0
	0 0	0 1 0 0 0 0 0 0	T dst	src1	src2
	Тур	be 2			
	31	24 23	161	5 8	7 0
	0 0	0 1 1 0 0 0 0 0	T dst	src1	src2
Word Fields					
Туре 1					
	Т	<i>src1</i> addressing r	modes	<i>src2</i> addressing) modes
	00) register mode (any	CPU register)	register mode (ar	ny CPU register)
	01	indirect mode (disp	o = 0, 1, IR0, IR1)	register mode (ar	ny CPU register)
	10) register mode (any	CPU register)	indirect mode (dis	sp = 0, 1, IR0, IR1)
	11	indirect mode (disp	o = 0, 1, IR0, IR1)	indirect mode (dis	sp = 0, 1, IR0, IR1)
Туре 2					
	Т	<i>src1</i> addressing m	nodes	src2 addressing	modes
	00	register mode (any	CPU register)	8-bit signed imme	diate
	01	register mode (any	CPU register)	indirect mode *+/ displacement)	ARn(5-bit unsigned
	10	indirect mode *+AF displacement)	Rn(5-bit unsigned	8-bit signed imme	diate
	11	indirect mode *+AR displacement)	In1(5-bit unsigned	indirect mode *+A displacement)	Rn2(5-bit unsigned
On and the s		1			

Operation

 $src1 + src2 + C \rightarrow dst$

Assembly Language Instructions 14-35

Description	The sum of the <i>src1</i> and <i>src2</i> operands and value of the C (carry) flag is loaded into the <i>dst</i> register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be signed integers.		
Status Bits	If ST (SET COND) = 0, the condition flags are modified if the destination register is $R0 - R11$. If ST (SET COND) = 1, they are modified for all destination registers.		
	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise U 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C 1 if a carry occurs, 0 otherwise 		
Mode Bit	OVM operation is affected by OVM bit value.		
Cycles	1		
Example	None		

Syntax	ADDF src, dst			
Operands	<i>src</i> : general-addressing modes <i>dst</i> : register (R0 – R11)			
Opcode				
	31 24	23	1615	870
	00000001	1 G	dst	SIC
Word Fields				
	•	G	src addressing modes	_
	•	00	register (R0-R11)	_
		01	direct	
		10	indirect	
		11	immediate	
	-			_
Operation	$dst + src \rightarrow dst$			
Description	The sum of the <i>dst</i> a <i>src</i> operands are as	and <i>src</i> ope ssumed to	rands is loaded into the <i>ds</i> be floating-point numbers	tregister. The <i>dst</i> and s.
Status Bits	LUF 1 if a floating	g-point und	erflow occurs, unchanged	d otherwise

LV	1 If a floating-point overflow occurs, unchanged otherwise
UF	1 if a floating-point underflow occurs, 0 otherwise
Ν	1 if a negative result is generated, 0 otherwise
Z	1 if a zero result is generated, 0 otherwise
V	1 if an floating-point overflow occurs, 0 otherwise
С	Unaffected

Mode Bit OVM operation is not affected by OVM bit value.

1

Cycles

ADDF Add Floating-Point Values

Example

ADDE	*AR4++(IR	1) , R5
	Before Ir	nstruction
AR4	80 9800h	
IR1	12Bh	66
R5	057980 0000h	6.23750e + 01
Data	at 80 9800h 86B 2800h	4.7031250e + 02
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

After Instruction

AR4	80 992Bhh	
IR1	12Bh	
R5	09052C 0000h	5.3268750e + 02

Data at 80 9800h 86B 2800h
4.7031250e + 02

LUF [0 0 LV

R5 [



0 Ν

0 Ζ

0 ٧

С 0

Syntax	ADD	DF3 si	rc2, src1,	dst					
Operands	src1 dst:	, src2:	type 1 o register	r type mode	2 three-oper (R0 – R11)	rand addressii	ng modes	6	
Opcode									
	Тур	e 1							
	31		24 23	3	161	5	87		0
	0 0	1 0 0	0 0 0 1	Т	dst	src1		src2	
	Тур	e 2							
	31		24 23	3	161	5	8 7		0
	0 0	1 1 0	0 0 0 1	Т	dst	src1		src2	
Word Fields									
Туре 1									
	Т	<i>src1</i> ad	dressing	mode	S	<i>src2</i> address	ing mode	S	
	00	register	mode (R0) – R11)	register mode	(R0 – R1	1)	
	01	indirect	mode (dis	p = 0,	1, IR0, IR1)	register mode	(R0 – R1	1)	
	10	register	mode (R0) – R11)	indirect mode	(disp = 0,	1, IR0, IR	1)
	11	indirect	mode (dis	p = 0,	1, IR0, IR1)	indirect mode	(disp = 0,	1, IR0, IR	1)
Type 2									
	Т	src1 ad	dressing	mode	S	src2 address	ing mode	S	
	01	register	mode (an	y CPU	register)	indirect mode	e *+ARn(5	-bit unsig	ned
									ام م ما
	11	indirect displace	mode *+A ement)	Rn1(5	-bit unsigned	indirect mode displacement	*+ARn2(5)	5-bit unsig	nea

Description	The sum of the <i>src1</i> and <i>src2</i> operands is <i>src2</i> , and <i>dst</i> operands are assumed to	is loaded into the <i>dst</i> register. The <i>src1,</i> o be floating-point numbers.
Status Bits	LUF1 if a floating-point underflow orLV1 if a floating-point overflow occUF1 if a floating-point underflow occUF1 if a negative result is generatedZ1 if a zero result is generated, 0V1 if an floating-point overflow occCUnaffected	ccurs, unchanged otherwise purs, unchanged otherwise ccurs, 0 otherwise ed, 0 otherwise otherwise ccurs, 0 otherwise
Mode Bit	OVM operation is not affected by OVM	l bit value.
Cycles	1	
Example	ADDF3 *+AR1(2),*+AR1(8),R4	
	Before Instruction	After Instruction
	AR1 2FF820h	AR1 2FF820h
	R4 0h	R4 070DB2 0000h 1.41695313e + 02
	Data at 22F F822h 700 F000h 1.28940e + 02	Data at 22F F828h 34C 2000h 1.27590e + 01
	LUF 0	LUF 0
	LV 0	LV 0
	UF 0	UF 0
	N 0	N0
	Z 0	Z 0
	V0	V0
	C 0	C 0

Syntax	ADDF3 STF	src2, src1 src3, dst2	, dst1						
Operands	src1: regi src2: indir dst1: regi src3: regi dst2: indir	ster (R0 – F ect (disp = ster (R0 – F ster (R0 – F ect (disp =	R7) 0, 1, l R7) R7) 0, 1, l	R0, IR1) R0, IR1)	•				
Opcode									
	31 1 1 0 0	24 1 1 0	23 dst1	16 src1	15	dst2	8 7	src2	0
Word Fields	None								
Operation	src1 + s $src3 \rightarrow c$	rc2 → dst1 dst2							
Description	A floating-p registers and This means and the ope ister, then S by the ADD	point additio re read at th s that if one eration bein DTF accepts DF3.	on and le begi e of the lg perfo s as inp	a floating nning ar paralle ormed in out the co	g-point st nd loaded l operati parallel ontents o	tore are per d at the end ons (STF) r (ADDF3) w f the registe	formed i of the ex reads fro rites to the er before	n parallel xecute cy om a regi he same i it is modi	. All rcle. ster reg- fied
	If src2 and	<i>dst2</i> point to	o the sa	ame loca	ation, <i>src</i>	2 is read be	fore the	write to a	lst2.
Status Bits	LUF 1 if a LV 1 if a UF 1 if a N 1 if a Z 1 if a V 1 if a C Unable	a floating-po a floating-po a floating-po a negative o a zero resu an floating-p ffected	oint ur oint ov oint ur result i It is ge point c	nderflow verflow c nderflow is gener enerated overflow	occurs, u occurs, u occurs, ated, 0 o , 0 other occurs, 0	unchanged nchanged o 0 otherwise therwise wise 0 otherwise	l otherwi otherwise e	se e	
Mode Bit	OVM opera	ation is not	affecte	ed by O	/M bit va	lue.			
Cycles	1								

ADDF3||STF Parallel ADDF3 and STF

Example

ADDF3 *+AR3(IR1),R2,R5 || STF R4,*AR2

	Before In	struction
AR3	80 9800h	
IR1	0A5h	
R2	070C80 0000h	1.4050e + 02
R5	0h	
R4	057B40 0000h	6.281250e + 01
AR2	80 98F3h	
Data	at 80 98A5h 733 C000h	1.79750e + 02
Data	at 80 98F3h	
	oh	
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

	After Ins	truction
AR3	80 9800h	
IR1	0A5h	
R2	070C80 0000h	1.4050e + 02
R5	082020 0000h	3.20250e + 02
R4	057B40 0000h	6.281250e + 01
AR2	80 98F3h	
Data	at 80 98A5h 733 C000h	1.79750e + 02
Data	at 80 98F3h	
	57B 4000h	6.28125e + 01
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

Syntax	ADDI src, dst			
Operands	<i>src:</i> general-ad <i>dst</i> : register (a	ddressing m any register	nodes in CPU primary-register fi	le)
Opcode				
	_31	24 23	1615	8 7 0
	0 0 0 0 0 0 1	0 0 G	dst	src
Word Fields				
		G	src addressing modes	_
		00	register (any register in CPU primary-register file)	
		01	direct	
		10	indirect	
		11	immediate	
Operation	$dst + src \rightarrow dst$			
Description	The sum of the <i>c</i> dst and src operations	dst and <i>src</i> o ands are as	operands is loaded into th sumed to be signed intege	e the <i>dst</i> register. The ers.
Status Bits	If ST (SET CONI flags are modified registers.	D) = 0 and th d. If ST (SET	he destination register is F F COND) = 1, they are mod	R0 – R11, the condition dified for all destination
	LUF Unaffected LV 1 if an inte UF 0 N 1 if a nega Z 1 if a zero V 1 if an inte C 1 if a carry	d eger overflow ative result i result is ge eger overflow y occurs, 0 d	w occurs, unchanged othe s generated, 0 otherwise enerated, 0 otherwise w occurs, 0 otherwise otherwise	erwise
Mode Bit	OVM operation is	s affected by	y OVM bit value.	

Cycles

None

ADDI Add Integer

Example

ADDI R3,R7

	Before Instruct	ion
R3	0FFFF FFCBh	-53
R7	35h	53
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

After Instruction

-53

 R3
 OFFFF FFCBh

 R7
 Oh

 LUF
 O

 LV
 O

 UF
 O

 N
 O

Z 1 V 0

C 0

Syntax	ADDI3 src2, src1, dst
Operands	src1, src2: type 1 or type 2 three-operand addressing modes
	dst: register mode (any register in CPU primary-register file)
Opcode	
-	
	21 24.22 1615 8.7 (
	0 0 1 0 0 0 1 0 T dst src1 src2
	Туре 2
	<u>31 24 23 1615 8 7 (</u>
	0 0 1 1 0 0 0 1 0 T dst src1 src2
Word Fields	
Type 1	
турет	T src1 addressing modes src2 addressing modes
	00 register mode (any CPU register) register mode (any CPU register)
	01 indirect mode (disp = 0, 1, IR0, IR1) register mode (any CPU register)
	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1)
	10register mode (any CPU register)indirect mode (disp = 0, 1, IR0, IR1)11indirect mode (disp = 0, 1, IR0, IR1)indirect mode (disp = 0, 1, IR0, IR1)
Type 2	10register mode (any CPU register)indirect mode (disp = 0, 1, IR0, IR1)11indirect mode (disp = 0, 1, IR0, IR1)indirect mode (disp = 0, 1, IR0, IR1)
Туре 2	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) T src1 addressing modes src2 addressing modes
Туре 2	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) T src1 addressing modes src2 addressing modes 00 register mode (any CPU register) 8-bit signed immediate
Туре 2	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 src1 addressing modes src2 addressing modes 00 register mode (any CPU register) 8-bit signed immediate 01 register mode (any CPU register) indirect mode *+ARn(5-bit unsigned displacement)
Туре 2	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (any CPU register) 8-bit signed immediate 11 register mode (any CPU register) 8-bit signed immediate 11 indirect mode *+ARn(5-bit unsigned displacement) 8-bit signed immediate
Туре 2	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1) 10 register mode (any CPU register) 8-bit signed immediate 10 indirect mode *+ARn(5-bit unsigned displacement) 8-bit signed immediate 11 indirect mode *+ARn1(5-bit unsigned displacement) 8-bit signed immediate

Description	The su <i>src2, a</i>	Im of the <i>src1</i> and <i>src2</i> operands is loaded into the <i>dst</i> register. The <i>src1,</i> and <i>dst</i> operands are assumed to be signed integers.
Status Bits	If ST (flags a registe	SET COND) = 0 and the destination register is $R0 - R11$, the condition are modified. If ST (SET COND) = 1, they are modified for all destination ers.
	LUF LV UF N Z V C	Unaffected 1 if an integer overflow occurs, unchanged otherwise 0 1 if a negative result is generated, 0 otherwise 1 if a zero result is generated, 0 otherwise 1 if an integer overflow occurs, 0 otherwise 1 if a carry occurs, 0 otherwise
Mode Bit	OVM o	operation is affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	ADD STI	13 src2, src3,	src1, dst1 dst2						
Operands	<i>src1:</i> r <i>src2:</i> in <i>dst1:</i> r <i>src3:</i> r <i>dst2:</i> in	egister (F ndirect (c egister (F egister (F ndirect (c	R0 – R7) lisp = 0, 1 R0 – R7) R0 – R7) lisp = 0, 1	, IR0, IR1 , IR0, IR1	1)				
Opcode									
	31	0 1 1 1	24 23	orot	16	15 dat2	8 7		0
		0 1 1 1	asti	SICI	SIC3	USI2		SIC2	
Word Fields	None								
Operation	src1 src3 -	+ src2 → → dst2	→ dst1						
Description	An integ are read means to operation STI acco ADDI3.	ger addition d at the l that if one on being p epts as i	on and an beginning of the par berformed nput the c	integer s and loac rallel ope in paralle contents o	tore are p led at th rations (el (ADDI3 of the reg	berformed i e end of th STI) reads f 3) writes to t gister befor	n para e exe rom a he sai re it is	allel. All regis cute cycle. register and me register, modified by	sters This d the then y the
	lf <i>src2</i> a	nd <i>dst2</i> p	oint to the	same loc	cation, <i>sr</i>	c2is read b	efore	the write to	dst2.
Status Bits	LUF (LV 1 UF (N 1 Z 1 V 1 C 1	Jnaffecte if an inte if a nega if a zero if an inte if a carr	d eger overf ative resu result is g eger overf y occurs,	low occu It is gene generate low occu 0 otherwi	rs, uncha rated, 0 d, 0 othe rs, 0 oth	anged othe otherwise rwise erwise	rwise		
Mode Bit	OVM op	peration i	s affected	by OVM	bit value	9.			
Cycles	1								

ADDI3||STI Parallel ADDI3 and STI

AR0

IR0

R5

R0 R3

AR7

LUF

LV

UF

Ν

Ζ

۷

С

Data at 80 983Bh

Data at 80 992Ch

Examp

ADDI3 *AR0--(IR0),R5,R0

	Before Instruction
STI	R3,*AR7

80 992Ch

0Ch

0h

35h

12Ch

0h

0

0

0

0

0

0

0

80 983Bh

0DCh

n	After Instruction
	AR0 80 9920h
	IR0 0Ch
220	R5 0DCh
	R0 208h
53	R3 35h
	AR7 80 983Bh
300	Data at 80 992Ch
	Data at 80 983Bh 35h

LUF

LV

300

220

520

53

53

5	J

35h	
0	
0	

UF	0
N	0

Ν	0
Z	0

- Ζ 0 ۷
- 0 С

Suntay		ere det						
Operands	AND src, dstsrc:general-addressing modesdst:register (any register in CPU primary-register file)							
Opcode								
	31		24 23		1615		8 7	0
	00	0 0 0	101	G	dst		src	
Word Fields								
				G	<i>src</i> address	ing modes		
				00	register (any CPU primary	register in r-register file)	-	
				01	direct			
				10	indirect			
				11	immediate		_	
Operation	dst A	ND <i>src</i> –	→ dst					
Description	The b <i>dst</i> re	oitwise-log gister. Th	gical AN ne <i>dst</i> a	ND betv nd <i>src</i>	ween the <i>dst</i> a operands are a	nd <i>src</i> operan assumed to b	ds is loaded in e unsigned inte	to the egers.
Status Bits	If ST flags regist	(SET CO are modif ers.	ND) = 0 fied. If S) and ti ST (SE ⁻	ne destination ΓCOND) = 1, t	register is R0 hey are modifi	– R11, the con ied for all destir	dition nation
	LUF LV UF N Z V C	Unaffec Unaffec 0 MSB of 1 if a ze 0 Unaffec	eted ted the out ero resu	put It is ge	nerated, 0 oth	erwise		
Mode Bit	OVM	operatio	n is not	affecte	ed by OVM bit	value.		
Cycles	1							

AND Bitwise Logical-AND

Example

AND	R1,R2
	Before Instruction
R1	80h
R2	0AFFh
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	1

After Instruction

R1	80h
R2	80h
LUF	0

- LV 0
- UF 0
- N 0
- Z 0
- V _____0

C 1

Syntax	AND	3 src2	, src1	, dst				
Operands	<i>src1, src2:</i> type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)							
Opcode								
	Туре	e 1						
	31			24 23	8	161	5	8 7 0
	0 0	1 0 0	0 0	1 1	Т	dst	src1	src2
	Туре	2						
	31			24 23	;	161	5	8 7 0
	0 0	1 1 0	0 0	1 1	Т	dst	src1	src2
Word Fields								
Туре 1								
	Т	src1 a	ddres	sing	mode	s	src2 addressing	g modes
	T 00	<i>src1</i> a registe	ddres er mod	sing e (any	mode: / CPU	s register)	src2 addressing register mode (a	g modes any CPU register)
	T 00 01	src1 a registe indirec	ddres er mod t mode	sing e (any e (disp	mode: / CPU p = 0,	s register) 1, IR0, IR1)	<i>src2</i> addressing register mode (a register mode (a	g modes any CPU register) any CPU register)
	T 00 01 10	src1 a registe indirec registe	ddres er mod t mode er mod	sing e (any e (disp e (any	mode: y CPU p = 0, y CPU	s register) 1, IR0, IR1) register)	<i>src2</i> addressing register mode (a register mode (a indirect mode (d	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1)
	T 00 01 10 11	src1 a registe indirec registe indirec	ddres er mod t mode er mod t mode	sing e (any e (disp e (any e (disp	mode: y CPU p = 0, y CPU p = 0,	s register) 1, IR0, IR1) register) 1, IR0, IR1)	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1)
Туре 2	T 00 01 10 11	src1 a registe indirec registe indirec	ddres er mod t mod er mod t mod	sing (e (any e (disp e (any e (disp	mode: y CPU p = 0, y CPU p = 0,	s register) 1, IR0, IR1) register) 1, IR0, IR1)	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1)
Туре 2	T 00 01 10 11	src1 a registe indirec registe indirec src1 a	ddres er mod t mode er mod t mode ddres	sing e (any e (disp e (any e (disp sing	modes y CPU o = 0, y CPU o = 0, modes	s register) 1, IR0, IR1) register) 1, IR0, IR1) s	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d src2 addressing	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1) g modes
Туре 2	T 00 01 10 11 T 00	src1 a registe indirec indirec src1 a registe	ddres er mod er mod t mode t mode ddres er mod	sing e (any e (disp e (any e (disp sing sing	mode: y CPU p = 0, y CPU p = 0, mode: y CPU	s register) 1, IR0, IR1) register) 1, IR0, IR1) s register)	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d src2 addressing 8-bit signed imm	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1) g modes nediate
Туре 2	T 00 01 10 11 T 00 01	src1 a registe indirec indirec indirec src1 a registe registe	ddres er mod t mode t mod t mod ddres er mod	sing e (any e (disp e (any e (disp sing e (any e (any	mode: y CPU p = 0, y CPU p = 0, mode: y CPU y CPU y CPU	s register) 1, IR0, IR1) register) 1, IR0, IR1) s register) register)	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d src2 addressing 8-bit signed imm indirect mode * displacement)	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1) g modes mediate r+ARn(5-bit unsigned
Туре 2	T 00 01 10 11 T 00 01 10	src1 a registe indirec indirec src1 a registe indirec displac	ddres er mod t mode t mode t mode er mod er mod t mod cemen	sing e (any e (disp e (any e (disp sing e (any le *+ <i>A</i> t)	mode: y CPU p = 0, y CPU p = 0, mode: y CPU y CPU ARn(5-	s register) 1, IR0, IR1) register) 1, IR0, IR1) s register) register) -bit unsigned	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d src2 addressing 8-bit signed imm indirect mode * displacement) 8-bit signed imm	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1) disp = 0, 1, IR0, IR1) g modes hediate r+ARn(5-bit unsigned hediate
Туре 2	T 00 01 10 11 T 00 01 10 11	src1 a registe indirec indirec src1 a registe registe indirec displac	ddres er mod t mode t mod t mod er mod t mod cemen t mod cemen	sing e (any e (disp e (any e (disp sing e (any e (any le *+A t)	mode: y CPU p = 0, y CPU p = 0, mode: y CPU y CPU y CPU ARn(5- Rn1(5-	s register) 1, IR0, IR1) register) 1, IR0, IR1) s register) register) -bit unsigned	src2 addressing register mode (a register mode (a indirect mode (d indirect mode (d src2 addressing 8-bit signed imm indirect mode * displacement) 8-bit signed imm indirect mode *+ displacement)	g modes any CPU register) any CPU register) lisp = 0, 1, IR0, IR1) lisp = 0, 1, IR0, IR1) g modes mediate f+ARn(5-bit unsigned mediate

Assembly Language Instructions 14-51

Description	The bitwise logical-AND between the <i>src1</i> and <i>src2</i> operands is loaded into the dst register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be unsigned integers. The immediate <i>src2</i> addressing mode is sign-extended.							
Status Bits	If ST (SET COND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.							
	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero result is generated, 0 otherwise V 0 C Unaffected 							
Mode Bit	OVM operation is not affected by OVM bit value.							
Cycles	1							
Example	Notice the difference between AND and AND3, in this example:							
	AND380h, R0, R0R0=FFFFFFFFhR0=FFFFFF80hAND80h, R0R0=FFFFFFFFhR0=00000080h							

Syntax	AND3 src2, src1, dst1 STI src3, dst2
Operands	src1: register (R0 - R7)src2: indirect (disp = 0, 1, IR0, IR1)dst1: register (R0 - R7)src3: register (R0 - R7)dst2: indirect (disp = 0, 1, IR0, IR1)
Opcode	
	<u>31 24 23 1615 8 7 0</u>
	1 1 0 1 0 0 0 dst1 src1 src3 dst2 src2
Word Fields	None
Operation	$src1 \text{ AND } src2 \rightarrow dst1$ $src3 \rightarrow dst2$
Description	A bitwise-logical AND and an integer store are performed in parallel. All regis- ers are read at the beginning and loaded at the end of the execute cycle. This neans that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (AND3) writes to the same register, then STI accepts as input the contents of the register before it is modified by the AND3.
	f <i>src2</i> and <i>dst2</i> point to the same location, <i>src2</i> is read before the write to <i>dst2</i> .
Status Bits	 UF Unaffected V Unaffected JF 0 N MSB of the output 1 if a zero result is generated, 0 otherwise 0 C Unaffected
Mode Bit	DVM operation is not affected by OVM bit value.
Cycles	

AND3||STI Parallel AND3 and STI

Example

AND3 *+AR1(IR0),R4,R7 || STI R3,*AR2

	Before I	nstruction
AR1	80 99F1h	
IR0	8h	
R4	0A323h	
R7	0h	
R3	35h	
AR2	80 983Fh	
Data	at 80 99F9h 5C53h	
Data	at 80 983Fh	
	0h	
LUF	0	
LV	0	
UF	0	
N	0	
Z	0	
V	0	
С	0	

After Instruction 80 99F1h AR1 IR0 8h 0A323h R4 R7 03h R3 35h AR2 80 983Fh Data <u>at 80 99F9h</u> 5C53h Data <u>at 80 983Fh</u> 35h

> 0 0

> 0

0 0

0

0

LUF

LV UF

Ν

Ζ

۷

С

53

53

53

Syntax	ANDN src, dst					
Operands	<i>src:</i> general-addressing modes <i>dst</i> : register (any register in CPU primary-register file)					
Opcode						
	31 24 2	3	1615	8 7 0		
	0 0 0 0 0 0 1 1 0	G	dst	STC		
Word Fields						
		G	<i>src</i> addressing modes			
		00	register (any register in CPU primary-register file)			
		01	direct			
		10	indirect			
		11	immediate			
Operation	$dst \text{ AND } \sim src \rightarrow dst$					
Description	The bitwise-logical Al plement (~) of the <i>sra</i> operands are assum	ND betw c operan ed to be	een the <i>dst</i> operand and th d is loaded into the <i>dst</i> reg unsigned integers.	e bitwise-logical com- jister. The <i>dst</i> and <i>src</i>		
Status Bits	If ST (SET COND) = flags are modified. If registers.	0 and th ST (SET	e destination register is R(COND) = 1, they are mod	0 – R11, the condition ified for all destination		
	LUFUnaffectedLVUnaffectedUF0NMSB of the outZ1 if a zero restV0CUnaffected	itput ult is gei	nerated, 0 otherwise			
Mode Bit	OVM operation is no	t affecte	d by OVM bit value.			
Cycles	1					

Example

ANDN @980Ch,R2

	Before Instruction		After Ins
DP	80h	DP	80h
R2	0C2Fh	R2	042Dh
Data	at 80 980Ch 0A02h	Data	at 80 980Ch 0A02h
LUF	0	LUF	0
LV	0	LV	0
UF	0	UF	0
Ν	0	Ν	0
Z	0	Z	0
V	0	V	0
С	0	С	0

After Instruction

Syntax	ANDN	13 src2, s	src1, ds	st			
Operands	src1, s	src2: ty	/pe 1 o	r type	2 three-oper	and addressing	nodes
	dst:	re	egister i	mode	(any register	r in CPU primar	y-register file)
Oncode							
Opcode							
	Туре	1					
	31		24 23	3	161	5	8 7 0
	001	0 0 0	1 0 0	Т	dst	src1	src2
	Type 2	2					
	31		24 23	3	161	5	8 7 0
	0 0 1	100	1 0 0	Т	dst	src1	src2
Word Fields							
Type 1							
турет	T	ro1 oddr	oooina	mode		ara? addroooin	a modoo
					S	sicz addressii	
	00 1			y CPU			any CPU register)
	10		ode (ais	p = 0,	I, INU, INI)	indirect mode (a	
	10 1	egister m		у С Г О ъ о		indirect mode (c	disp = 0, 1, 100, 101
		iurect m	ue (uis	ρ = 0,	1, INU, INI)		usp = 0, 1, 10, 11)
Туре 2							
	T s	<i>rc1</i> addr	essing	mode	s	src2 addressin	g modes
	00 r	egister m	ode (an	y CPU	register)	8-bit signed imn	nediate
	01 r	eaister m	ode (an	v CPU	register)	indirect mode	*+ARn(5-bit unsigned
		- 3		,	rogiotor)	displacement)	
	10 ^{ir} d	ndirect m lisplacem	ode (un ode *+/ ent)	ARn(5	-bit unsigned	displacement) 8-bit signed imn	nediate
	10 d 11 d	ndirect m lisplacem ndirect m lisplacem	ode (un ode *+/ ent) ode *+A ent)	ARn(5 .Rn1(5	-bit unsigned	alsplacement) 8-bit signed imm indirect mode *- displacement)	nediate +ARn2(5-bit unsigned

Description	The bitwise-logical AND between the <i>src1</i> operand and the bitwise-logical complement (~) of the <i>src2</i> operand is loaded into the <i>dst</i> register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be unsigned integers. The immediate <i>src2</i> addressing mode is sign-extended.			
Status Bits	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero result is generated, 0 otherwise V 0 C Unaffected 			
Mode Bit	OVM operation is not affected by OVM bit value.			
Cycles	1			
Example	None			

Syntax	ASH src_c	count, dst			
Operands	<i>src_count:</i> general-addressing modes <i>dst:</i> register (any register in CPU primary-register file)				
Opcode					
	31	24 23	1615	8 7 0	
	00000	0111 G	dst	src_count	

Word Fields

G	<i>src</i> addressing modes
00	register (any register in CPU primary-register file)
01	direct
10	indirect
11	immediate

Operation	<i>count =</i> 7 LSBs of <i>src_count</i>
	If $(count \ge 0)$:
	$dst << count \rightarrow dst$
	Else:
	$dst >> count \rightarrow dst$

Description The seven least-significant bits of the *src_count* operand constitute the 2s-complement shift *count* of up to 32 bits.

If *count* is greater than 0, the *dst* operand is left-shifted by the value of *count*. Low-order bits shifted in are zero-filled, and high-order bits are shifted out through the C (carry) bit.

Arithmetic left-shift:



If *count* is less than 0, the *dst* operand is right-shifted by the absolute value of *count*. The high-order bits of the *dst* operand are sign-extended as it is right-shifted. Low-order bits are shifted out through the C (carry) bit.

Arithmetic right-shift:



Assembly Language Instructions 14-59

If <i>count</i> is 0, no shift is <i>src_count</i> and <i>dst</i> operar	performed, ids are assu	and the C (carry) bit is set med to be signed integers.	to 0. The			
If ST (SET COND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.						
LUFUnaffectedLV1 if an integer overUF0NMSB of the outputZ1 if a zero result isV1 if an integer overCSet to the value of	flow occurs, generated, flow occurs, the last bit s	, unchanged otherwise 0 otherwise , 0 otherwise shifted out. 0 for a shift <i>coun</i>	<i>t</i> of 0			
OVM operation is not affe	ected by OVI	M bit value.				
1						
ASH R1,R3 Before Instruct R1 10h R3 0A E000h LUF 0 LV 0 UF 0 UF 0 N 0 Z 0 V 0 C 0	ion 16	After Instruction R1 10h R3 0E000 0000h LUF 0 LV 1 UF 0 N 1 Z 0 V 1 C 0	n			
ASH @98C3h,R5 Before Instruct DP 80h R5 0AEC0 0001h Data at 80 98C3h 0FFE8 LUF 0 LV 0 UF 0 UF 0 N 0 Z 0 V 0 C 0	ion 16 –24	After Instruction DP 80h R5 0FFFF FFAEh Data at 80 98C3h 0FFE8 LUF 0 LV 0 UF 0 N 1 Z 0 V 0 C 1	n —24			
	If count is 0, no shift is src_count and dst operaries if ST (SET COND) = 0 and flags are modified. If ST (Sergisters. LUF Unaffected LV 1 if an integer over UF 0 N MSB of the output Z 1 if a zero result is V 1 if an integer over C Set to the value of OVM operation is not affected 1 ASH R1, R3 Before Instruct R1 10h R3 0A E000h LUF 0 LV 0 UF 0 N 0 Z 0 V 0 C 0 ASH @98C3h, R5 Before Instruct DP 80h R5 0AEC0 0001h Data at 80 98C3h R5 0AEC0 0001h Data at 80 98C3h R5 0AEC0 0001h Data at 80 98C3h N 0 Z 0 V 0 UF 0 LUF 0 LUF 0 LUF 0 LUF 0 LUF 0 C 0 C 0 N 0 C 0 C 0 C 0 C 0 C 0 C 0 C 0 C	If count is 0, no shift is performed, src_count and dst operands are assured that the structure of the destination of the second seco	If count is 0, no shift is performed, and the C (carry) bit is set src_count and dst operands are assumed to be signed integers. If ST (SET COND) = 0 and the destination register is R0 – R11, th flags are modified. If ST (SET COND) = 1, they are modified for all registers. LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N MSB of the output Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C Set to the value of the last bit shifted out. 0 for a shift count OVM operation is not affected by OVM bit value. 1 Ash R1, R3 Meter Instruction R1 10h R3 0AE000h R4 0 UF 0			

Syntax	ASH3 src_count, src, dst				
Operands	src, src_counttype 1 or type 2 three-operand addressing modesdstregister mode (any register in CPU primary register file)				
Opcode					
	Туре 1				
	31 24 23 1615 8 7 <u>0</u>				
	0 0 1 0 0 0 1 0 1 T dst src src_count				
	Туре 2				
	<u>31 24 23 1615 8 7 0</u>				
	0 0 1 1 0 0 1 0 1 T dst src src_count				
Word Fields					
Туре 1					
	T src1 addressing modes src2 addressing modes				
	00 register mode (any CPU register) register mode (any CPU register)				
	01 indirect mode (disp = 0, 1, IR0, IR1) register mode (any CPU register)				
	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1)				
	11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1)				
Туре 2					
	T src1 addressing modes src2 addressing modes				
	00 register mode (any CPU register) 8-bit signed immediate				
	01 register mode (any CPU register) indirect mode *+ARn(5-bit unsigned displacement)				
	10 indirect mode *+ARn(5-bit unsigned displacement) 8-bit signed immediate				
	11 indirect mode *+ARn1(5-bit unsigned displacement) indirect mode *+ARn2(5-bit unsigned displacement)				
Operation	$count = 7 \text{ LSBs of } src_count$ if $(count \ge 0)$ $src < count \rightarrow dst$				
	$src > > count \rightarrow dst$				

Description The seven least-significant bits of the *src_count* operand constitute the 2s-complement shift *count*.

If *count* is greater than 0, the *src* operand is left-shifted by the value of *count*. Low-order bits shifted in are zero-filled, and high-order bits are shifted out through the status register's C (carry) bit.

Arithmetic left-shift:



If *count* is less than 0, the *src* operand is right-shifted by the absolute value of *count* (e.g. -4 = right-shift 4). The high-order bits of the *src* operand are sign-extended as they are right-shifted. Low-order bits are shifted out through the C (carry) bit.

Arithmetic right-shift:



If *count* is 0, no shift is performed, and the C (carry) bit is set to 0. The *src_count*, *src*, and *dst* operands are assumed to be signed integers.

Status Bits	LUF LV UF N Z V C	Unaffected 1 if an integer overflow occurs, unchanged otherwise 0 MSB of the output 1 if a zero result is generated, 0 otherwise 1 if an integer overflow occurs, 0 otherwise Set to the value of the last bit shifted out. 0 for a shift <i>count</i> of 0
Mode Bit	OVM	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	ASH3 ∥ STI	src_count, src src3, dst2	c2, dst1				
	$src2 << count \rightarrow dst1$						
	Else: src2 >> cc $ src3 \rightarrow a$	ount $ ightarrow dst1$ dst2					
Operands	src_count	register (R0 – F	R7)				
	<i>src2:</i> indir <i>dst1:</i> regis <i>src3:</i> regis <i>dst2:</i> indir	rect (disp = 0, 1 ister ($R0 - R7$) ister ($R0 - R7$) rect (disp = 0, 1	, IR0, IR1) , IR0, IR1))			
Opcode							
	31	24 23	1	1615	8	7 0	, 1
	1 1 0 1 0	0 0 1 dst1	src_count	src3	dst2	src2	J
Word Fields	None						
Operation	<i>count</i> = 7 L If (<i>count</i> ≥	LSBs of <i>src_cou</i> 0):	unt				
Description	The seven the 2s-com	n least-significar nplement shift <i>c</i>	nt bits of th <i>ount</i> of up	ne <i>src_co</i> to 32 bits	<i>unt</i> operand i S.	register constitute	Э
	If <i>count</i> is g Low-order through the	greater than 0, t bits shifted in a e C (carry) bit.	he <i>dst</i> ope are zero-fi	erand is le illed, and	ft-shifted by t high-order b	he value of <i>count</i> its are shifted ou	t. it
	Arithmetic	left-shift:					
		С	•	scr2	0		
	If <i>count</i> is less than 0, the <i>dst</i> operand is right-shifted by the absolute value of <i>count</i> . The high-order bits of the <i>dst</i> operand are sign-extended as it is right-shifted. Low-order bits are shifted out through the C (carry) bit.)f (-	
	Arithmetic	right-shift:					
		sign of <i>src.</i>	2	S	rc2	→ C	

If *count* is 0, no shift is performed, and the C (carry) bit is set to 0. The *src_count* and *dst* operands are assumed to be signed integers.
	All registers are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (ASH3) writes to the same register, then STI accepts as input the contents of the register before it is modified by the ASH3. If <i>src2</i> and <i>dst2</i> point to the same location, <i>src2</i> is read before the write to <i>dst2</i> .							
Status Bits	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N MSB of the output Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C Set to the value of the last bit shifted out. 0 for a shift count of 0 							
Mode Bit	ΟνΜ	operation is not affe	cted by OVM	bit val	ue.			
Cycles	1							
Example	ASI STI	H3 R1,*AR6++(I I R5,*AR2	R1),R0					
		Before Instruction	on		After Instru	uction		
		20.0.0	• • •					
	AR6	80 9900h		AR6	80 998Ch			
	AR6	80 9900h 8Ch		AR6 IR1	80 998Ch 8Ch			
	AR6 [R1 [R1 [80 9900h 8Ch 0FFE8h	-24	AR6 IR1 R1	80 998Ch 8Ch 0FFE8h	-24		
	AR6 [R1 [R1 [R0 [80 9900h 8Ch 0FFE8h 0h	-24	AR6 IR1 R1 R0	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh	-24		
	AR6 [R1 [R1 [R0 [R5 [80 9900h 8Ch 0FFE8h 0h 35h	-24 53	AR6 IR1 R1 R0 R5	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h	24 53		
	AR6 [IR1 [R1 [R0 [AR2 [Data a	80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h t 80 9900h	-24 53	AR6 IR1 R1 R0 R5 AR2 Data	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h	-24 53		
	AR6 [R1 [R0 [R5 [AR2 [Data a	80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h t 80 9900h 0AE00 0000h	24 53	AR6 IR1 R1 R0 R5 AR2 Data	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h 0AE00 0000h	24 53		
	AR6 [IR1 [R1 [R0 [R5 [Data a Data a	80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h 0AE00 0000h t 80 98A2h 0AE00 0000h	-24 53	AR6 IR1 R0 R5 AR2 Data Data	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h 0AE00 0000h at 80 98A2h 35h	24 53 53		
	AR6 [IR1 [R1 [R0 [R5 [AR2 [Data a [Data a [LUF [80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h t 80 9900h 0AE00 0000h t 80 98A2h 0h 0	-24 53	AR6 IR1 R0 R5 AR2 Data Data	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h 0AE00 0000h at 80 98A2h 35h 0	24 53 53		
	AR6 [IR1 [R1 [R5 [Data a Data a LUF [LV [80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h 0AE00 0000h t 80 98A2h 0AE00 0000h t 80 98A2h 0h 0	24 53	AR6 IR1 R0 R5 AR2 Data Data LUF LV	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h 0AE00 0000h at 80 98A2h 35h 0 35h 0 35h 0 35h 0 35h 0 0 0 0 0	-24 53 53		
	AR6 [IR1 [R1 [R0 [R5 [AR2 [Data a Data a LUF [LV [UF [80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h t 80 9900h 0AE00 0000h t 80 98A2h 0h 0AE00 0000h t 80 98A2h 0h 0	-24 53	AR6 IR1 R0 R5 AR2 Data Data LUF LV UF	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h 0AE00 0000h at 80 98A2h 35h 0 0 0 0 0	-24 53 53		
	AR6 [IR1 [R1 [R0 [R5 [AR2 [Data a Data a LUF [LV [N [N [N [N []	80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h 0AE00 0000h t 80 98A2h 0h 0AE00 0000h t 80 98A2h 0h 0 0 0 0 0 0 0 0 0 0 0 0	24 53	AR6 IR1 R0 R5 AR2 Data Data LUF LV UF N	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h 0AE00 0000h at 80 98A2h 35h 0 0 0 0 0 1	-24 53 53		
	AR6 [IR1 [R1 [R0 [R5 [AR2 [Data a Data a LUF [LV [UF [N [Z [X [80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h t 80 9900h 0AE00 0000h t 80 98A2h 0h 00 00 00 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 000 01	-24 53	AR6 IR1 R0 R5 AR2 Data Data LUF LV UF N Z	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h 0AE00 0000h at 80 98A2h 35h 0 0 0 1 0	24 53 53		
	AR6 [IR1 [R1 [R0 [R5 [AR2 [Data a LUF [LUF [LV [UF [N [Z [V [V [V [V [V [V [V [V	80 9900h 8Ch 0FFE8h 0h 35h 80 98A2h t 80 9900h 0AE00 0000h t 80 98A2h 0h 00 00 00 00 01 02 03 04 05 06 07 08 0900h 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 00 00	24 53	AR6 IR1 R0 R5 AR2 Data Data LUF LV UF N Z V	80 998Ch 8Ch 0FFE8h 0FFFF FFAEh 35h 80 98A2h at 80 9900h 0AE00 0000h at 80 98A2h 35h 0 0 0 0 0 0 0 0 0 0 0 0 0	-24 53 53		

Syntax	Bcond src							
Operands	src: conditional-branch-addressing modes (B)							
Opcode								
	<u>31 24 23 1615 8 7 0</u>							
	0 1 1 0 1 0 B 0 0 0 0 <i>cond</i> register or <i>displacement</i>							
Word Fields								
	B src addressing modes							
	0 register							
	1 PC relative							
Operation	If <i>cond</i> is true: If <i>src</i> is in register-addressing mode (any register in CPU primary- register file), <i>src</i> \rightarrow PC. If <i>src</i> is in PC-relative mode (label or address), <i>displacement</i> + PC + 1 \rightarrow PC. Else, continue.							
Description	Bcond signifies a standard branch that executes in four cycles. A branch is performed if the condition is true (since a pipeline flush also occurs on a true condition; see Section 8.2 on page 8-4). If the <i>src</i> operand is expressed in register- addressing mode, the contents of the specified register are loaded into the PC. If the <i>src</i> operand is expressed in PC-relative mode, the assembler generates a displacement: <i>displacement</i> = label – (PC of branch instruction + 1). This dis- placement is stored as a 16-bit signed integer in the 16 least-significant bits of the branch instruction word. This displacement is added to the PC of the branch instruction plus 1 to generate the new PC.							
	The 'C4x provides 20 condition codes that can be used with this instruction (see Section 14.2 for a list of condition mnemonics, encoding, and flags).							
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected							
Mode Bit	OVM operation is not affected by OVM bit value.							
Cycles	4 (Regardless of whether or not the branch is taken)							

Example

ΒZ	R0					
		Before In	struction		After Ins	truction
PC		2B00h		PC	3FF00h	
R0		0003 FF00h		R0	0003 FF00h	
LUF	-	0		LUF	0	
LV		0		LV	0	
UF		0		UF	0	
Ν		0		Ν	0	
Ζ		1		Z	1	
V		0		V	0	
С		0		С	0	

Note:

If a BZ instruction is executed immediately following a RND instruction with a zero operand, the branch is not performed, because the zero flag is not set. To circumvent this problem, execute a BZUF instead of a BZ instruction.

Syntax	BcondAF src							
Operands	src: conditional-branch-addressing modes							
Opcode								
	31 24	23	1615		8 7	0		
	0 1 1 0 1 0 B 0	101	cond	register	or displacement			
Word Fields								
	-	В	src addres	sing modes	_			
	-	0	register					
		1	PC relative					
Operation	If (<i>cond</i> is true) If (<i>src</i> is a registe	r)						
	$src \rightarrow PC$ If (src is in PC-rel	ative mo	nde)					
	displacement	PC of b	pranch + 3 \rightarrow	PC				
	Else: If (<i>cond</i> is false)							
	annul the effect of effect of the read instructions and c	the exec and exe continue.	cute phase of cute phases	the first follow of the second	ving instruction an d and third followi	d the ing		
Description	If the condition is true instruction are execu- the effect of the execu- and execute phases The three instruction and is in register mo- into the PC. If the sro of the branch instruct relative mode the dis	e, a bran uted. If th cute phas of the se s followin de, then c operan tion + 3 a splacem	ch and the thr ne condition is se of the first econd and thin ng B <i>cond</i> AF of the contents id is in PC-rel- and the <i>displat</i> <i>ent</i> field is int	ree instruction a false, no bra following inst rd following ir do not affect th of the specifi ative mode, t <i>cement</i> is loa erpreted as a	ns following the br anch is performed ruction and of the nstructions is annu- ne <i>cond</i> . If the <i>src</i> ied register are lo hen the sum of th ded into the PC. Ir a 16-bit signed int	anch , and read ulled. oper- aded e PC n PC- eger.		
	None of the three ins modifies the progra B <i>cond</i> AF instruction	tructions m flow. 1.	s following the Interrupts ar	B <i>cond</i> AF ca e disabled fo	n be an instruction or the duration o	n that If the		
	B <i>cond</i> AF especially caution when using modify registers like also applies when us modification.	is useful instructio AR <i>n</i> , SI sing inst	for controlling ons such as F P, and DP in t ructions to pe	g the exit at th PUSH/POP, L he decode ar erform indirec	e bottom of a loop .DPK, or LDA tha nd/or read phase. t addressing with	. Use t can This AR <i>n</i>		

Status Bits	LUF	Unaffected
	LV	Unaffected
	UF	Unaffected
	Ν	Unaffected
	Z	Unaffected
	V	Unaffected
	С	Unaffected
Mode Bit	OVM	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	BcondAT src								
Operands	<i>src</i> conditional-b	src conditional-branch-addressing modes							
Opcode									
	31	24 23	1615		8 7	0			
	0 1 1 0 1 0 E	B 0 0 1 1	cond	register o	or displacement				
Word Fields									
		В	<i>src</i> addressir	ng modes					
		0	register		_				
		1	PC relative						
Operation	If (<i>cond</i> is true) If (<i>src</i> is a regist $src \rightarrow PC$ annul the effe the read and If (<i>src</i> is in PC-re <i>displacement</i> annul the effe the read and Else, continue.	er) ect of the exe execute pha elative mode t + PC of bra ect of the exe execute pha	ecute phase of the ases of second a se	ne first follow and third fol ne first follow and third fol	ving instruction a lowing instruction ving instruction a lowing instructio	and of ons. and of ons.			
Description	If the condition of phase of the firs second and thir lowing B <i>cond</i> AT ter mode, then t If the <i>src</i> operan instruction + 3 a mode, the <i>displa</i> None of the thre modifies the pro The B <i>cond</i> AT ins faces. Be espe- LDPK, or LDA th and/or read phase	Is true, a bra t following ir d following i do not affec he contents d is in PC-re and the <i>disp</i> acement fiel ee instructio gram flow. Ir struction doe cially carefu hat can moo se. This also	anch is performent instruction and of instructions is an of the specified lative mode, the placement are lo d is interpreted ins following Bcd interrupts are disa es not annul the s il when using in lify registers like applies when yo	ed, and the f the read ar nulled. The src operand I register are nothe sum of aded into the as a 16-bit s ondAT can h abled for the status signal nstructions ARn, SP, a ou use instru	effect of the exi nd execute phas three instruction is expressed in the e loaded into the f the PC of the bi- ne PC. In PC-re signed integer. be an instruction e duration of Bco ls at the external such as PUSH/ and DP in the de- ctions to perform	ecute ses of ns fol- regis- e PC. ranch lative n that ndAT. inter- (POP, ecode n indi-			
	trolling the entry	at the top c	of the loop.	iur particu	nariy is userui 101	COII-			

Status Bits	LUF	Unaffected
	LV	Unaffected
	UF	Unaffected
	Ν	Unaffected
	Z	Unaffected
	V	Unaffected
	С	Unaffected
Mode Bit	ονμ	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax Operands	B <i>cond</i> D <i>src</i> <i>src:</i> conditional-branch-addressing modes (B)						
Opcode	31 24 23 1615 8 7 0						
	0 1 1 0 1 0 B 0 0 0 1 <i>cond</i> register or <i>displacement</i>						
Word Fields							
	B <i>src</i> addressing modes						
	0 register						
	1 PC relative						
Operation	If <i>cond</i> is true: If <i>src</i> is in register-addressing mode (any register in CPU primary- register file) <i>src</i> \rightarrow PC. If <i>src</i> in PC-relative mode (label or address), <i>displacement</i> + PC + 3 \rightarrow PC. Else, continue.						
Description	B <i>cond</i> D signifies a delayed branch, allowing the three instructions after the delayed branch to be performed before the PC is modified. The effect is a single-cycle branch, and the three instructions following B <i>cond</i> D do not affect the <i>cond</i> .						
	None of the three instructions following B <i>cond</i> D should be an instruction that modifies program flow. Interrupts are disabled for the duration of B <i>cond</i> D.						
	A branch is performed if the condition is true. If the <i>src</i> operand is expressed in register-addressing mode, the contents of the specified register are loaded into the PC. If the <i>src</i> operand is expressed in PC-relative mode, the assembler generates a displacement: <i>displacement</i> = <i>src</i> – (PC of branch instruction + 3). This displacement is stored as a 16-bit signed integer in the 16 least-signifi- cant bits of the branch instruction. This displacement is added to the PC of the branch instruction plus 3 to generate the new PC. The 'C4x provides 20 condi- tion codes that can be used with this instruction (see Section 14.2 for a list of condition mnemonics, encoding, and flags).						
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected						

Mode Bit	OVM operation is not affected by OVM b	pit value.
Cycles	1	
Example	BNZD 36 (36 = 24h)	
	Before Instruction	After Instruction
	PC 50h	PC 77h
	LUF 0	LUF 0
	LV 0	LV 0
	UF 0	UF 0
	N 0	N 0
	Ζ 1	Z 1
	V 0	V 0
	C 0	C 0

Syntax	BR src
Operands	src: in PC-relative mode
Opcode	
	<u>31 24 23 1615 8 7 0</u>
	0 1 1 0 0 0 0 0 0 displacement
Word Fields	None
Operation	$PC + 1 + displacement \rightarrow PC$
Description	Performs an unconditional branch. The assembler generates a displacement: displacement = src - (PC of branch instruction + 1). This displacement is stored as a 24-bit signed integer in the 24 least-significant bits of the branch nstruction. This displacement is added to the PC of the branch instruction plus I to generate the new PC.
Status Bits	UFUnaffectedVUnaffectedJFUnaffectedVUnaffectedZUnaffectedCUnaffected
Mode Bit	DVM operation is not affected by OVM bit value.
Cycles	4
Example	None

Syntax	BRD src							
Operands	<i>src</i> : in PC-relative mode							
Opcode								
	31 24 23 1615 8 7 0							
Word Fields	None							
Operation	$PC + 3 + displacement \rightarrow PC$							
Description	Performs an unconditional delayed branch. The assembler generates a displacement: $displacement = src - (PC of branch instruction + 3)$. This displacement is stored as a 24-bit signed integer in the 24 least significant bits of the branch instruction. This displacement is added to the PC of the branch instruction plus 3 to generate the new PC. Interrupts are disabled during the BRD instruction.							
	The three instructions following the BRD instruction are fetched and executed. None of these three instructions should modify the program flow (e.g., affect the PC value).							
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected							
Mode Bit	OVM operation is not affected by OVM bit value.							
Cycles	1							
Example	None							

Syntax	CALL	src						
Operands	<i>src</i> : in PC-relative mode							
Opcode								
	31		24 2	23	1615		8 7	0
	011	0001	0			displacement		
Word Fields	None							
Operation	Next F PC +	$PC \rightarrow *(++)$ 1 + <i>displac</i>	SP) <i>eme</i>	$nt \rightarrow PC$				
Description	Perfor sembl structi least s the PC	ms a call. er generat on + 1). Th significant I C of the bra	The r es a iis dis bits c anch	next PC va displacem splacemen of the brand instruction	ue is pus ent: <i>displ</i> t is stored ch instruct plus 1 to	hed onto the acement = s as a 24-bit s tion. This dis generate the	system stack. Th rc – (PC of bran igned integer in t placement is add e new PC.	ne as- ch in- he 24 ded to
Status Bits	LUF LV UF N Z V C	Unaffecte Unaffecte Unaffecte Unaffecte Unaffecte Unaffecte	d d d d d					
Mode Bit	OVM	operation i	s not	affected b	y OVM bi	t value.		
Cycles	4							
Example	None							

CALLcond Call Subroutine Conditionally

Syntax	CALLcond src					
Operands	<i>src:</i> conditional-branch-addressing modes (B)					
Opcode						
	31	24 23	1615		8 7	0
	0 1 1 1 0 0 B	0 0 0 0	cond	register o	r displacement	
Word Fields						
		В	<i>src</i> address	ing modes	-	
		0	register		_	
		1	PC relative			
					_	
Operation	If cond is true: Next PC \rightarrow *++ If src is in re register file), src \rightarrow PC. If src in PC-rela Else, continue.	-SP gister-addr tive mode (essing mode label or addre	e (any registe ess), <i>displacer</i>	er in CPU prim <i>ment</i> +PC+1→	nary- PC.
Description	A call is performed if the condition is true. If the condition is true, the next PC value is pushed onto the system stack. If the <i>src</i> operand is expressed in register-addressing mode, the contents of the specified register are loaded into the PC. If the <i>src</i> operand is expressed in PC-relative mode, the assembler generates a displacement: <i>displacement</i> = label – (PC of call instruction + 1). This displacement is stored as a 16-bit signed integer in the 16 least-significant bits of the call instruction word. This displacement is added to the PC of the call instruction plus 1 to generate the new PC.					
	(see Section 14.2	for a list of	condition mn	iemonics, enc	oding, and flags).
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected					
Mode Bit	OVM operation is	not affecte	d by OVM bit	value.		
Cycles	5 (Regardless of v	whether the	e condition is	true or not)		

Example

CALLNZ R5

	Before Instruction
PC	123h
SP	80 9835h
R5	789h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

	After Instruction		
PC	789h		
SP	80 9836h		
R5	789h		
Data at 9836h			
	124h		

LUF

- 0 0 LV
- 0 UF
- 0 Ν

Ζ 0

0 ٧

0 С

Syntax	CMPF src, dst	
Operands	src: dst:	general-addressing modes (G): register (R0 – R11)
Opcode		

31	24 2	3	16	15 8 7	0
000	001000	G	dst	src	

Word Fields

G	src addressing modes
00	register (R0 – R11)
01	direct
10	indirect
11	immediate

Operation	dst – src		
Description	The <i>src</i> operand is subtracted from the <i>dst</i> operand. The result is not loaded into any register; this allows for nondestructive compares. The <i>dst</i> and <i>src</i> operands are assumed to be floating-point numbers.		
Status Bits	 LUF 1 if a floating-point underflow occurs, unchanged otherwise LV 1 if a floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise C Unaffected 		
Mode Bit	OVM operation is not affected by OVM bit value.		
Cycles	1		

CMPF *+AR4,R6

	TAR4, KO	
	Before In	struction
AR4	80 98F2h	
R6	070C80 0000h	1.4050e + 02
Data	at 80 98F3h	
	070C 8000h	1.4050e + 02
LUF	0	
LV	0	
UF	0	
Ν	1	
Z	0	
V	0	
С	0	

	After Inst	ruction
AR4	80 98F2h	
R6	070C80 0000h	1.4050e + 02
Data	at 80 98F3h	
	070C 8000h	1.4050e + 02
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

Syntax	CMPF3 src2, src1			
Operands	<i>src1 – src</i> 2 type 1 or type 2 three-operand addressing modes			
Opcode				
-	Type 1			
	21 24.22	1615 9 7 0		
	0 0 1 0 0 0 1 1 0 T 0 0 0 0 0	src1 $src2$		
	Туре 2			
	31 24 23	<u>1615 8 7 0</u>		
	0 0 1 1 0 0 1 1 0 T 0 0 0 0 0	src1 src2		
Word Fields				
Туре 1				
	T src1 addressing modes	src2 addressing modes		
	00 register mode (R0 — R11)	register mode (R0 — R11)		
	01 indirect mode (disp = 0, 1, IR0, IR1)	register mode (R0 — R11)		
	10 register mode (R0 — R11)	indirect mode (disp = 0, 1, IR0, IR1)		
	11 indirect mode (disp = 0, 1, IR0, IR1)	indirect mode (disp = 0, 1, IR0, IR1)		
Type 2				
.)	T <i>src1</i> addressing modes	src2 addressing modes		
	01 register mode (R11–R0)	indirect mode *+ARn(5-bit unsigned displacement)		
	11 indirect mode *+ARn1(5-bit unsigned displacement)	indirect mode *+ARn2(5-bit unsigned displacement)		
Operation	src1 – src2			
Description	The <i>src2</i> operand is subtracted from the s into any register. This allows for nondest operands are assumed to be floating-po	<i>src1</i> operand. The result is not loaded ructive compares. The <i>src1</i> and <i>src2</i> wint numbers.		
Status Bits	 LUF 1 if a floating-point underflow occu LV 1 if a floating-point overflow occu UF 1 if a floating-point underflow occu N 1 if a floating-point underflow occu N 1 if a negative result is generated, 0 occu Z 1 if a zero result is generated, 0 occu V 1 if a floating-point overflow occu C Unaffected 	eurs, unchanged otherwise rs, unchanged otherwise eurs, 0 otherwise d, 0 otherwise otherwise rs, 0 otherwise		
Mode Bit	OVM operation is not affected by OVM	oit value.		
14-80				

Cycles

1

Example

CMPF3 *AR2,*AR3-(1)			
Before Instruction	After Instruction		
AR2 809831h	AR2 809831h		
AR3 809852h	AR3 809851h (decrement)		
Data <u>at 809831h</u> 77A7000h 2.5044e + 02	Data at 809831h 77A7000h 2.5044e + 02		
Data <u>at 809852h</u> 57A2000h 6.253125e + 01	Data at 809852h 57A2000h 6.253125e + 01		
LUF 0	LUF 0		
LV 0	LV 0		
UF 0	UF 0		
N0	N 1		
Z0	Z 0		
V0	V 0		
C 0	C 0		

Syntax	СМРІ	src, dst						
Operands	<i>src:</i> general-addressing modes <i>dst:</i> register (any register in CPU primary-register file)							
Opcode								
	31	24	23	161	15	8 7 0		
	000	00100	1 G	dst		SrC	J	
Word Fields								
		-	G	src ad	Idressing modes	-		
		-	00	registe CPU p	er (any register in primary-register file)	-		
			01	direct				
			10	indired	st			
		_	11	immeo	diate	_		
Operation	dst – s	src						
Description	The <i>si</i> into ar erands	c operand is s by register; this are assume	subtracte s allows f d to be si	ed from t or nonde igned int	he <i>dst</i> operand. Th estructive compares tegers.	e result is not loaded s. The <i>dst</i> and <i>src</i> op	k -	
Status Bits	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C 1 if a borrow occurs, 0 otherwise 							
Mode Bit	OVM o	operation is no	ot affecte	ed by O∖	/M bit value.			
Cycles	1							

Example

CMPI R3,R7

	Before Instruction
R3	898h
R7	3E8h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

2200

1000

R3

R7

LUF

LV

UF

N Z

٧

С

After Instruction 898h

0

0

1

0

0

0

3E8h

2200

1000

Syntax	CMPI	3 src2, src	1					
Operands	src1 -	- <i>src</i> 2 type	1 or type 2	2 three-ope	erand ac	dressing i	modes	
Opcode								
	Туре	1						
	31		24 23		1615		8 7	0
	001	0001	1 1 T	0000	0 0	src1		src2
	Туре	2						
	31		24 23		1615		8 7	0
	001	1001	1 1 T	0000	0 0	src1		src2
Word Fields								
Туре1								
	Т	src1 addres	sing mode	S	src2	addressin	ng modes	
	00 r	register mod	e (any CPL	l register)	regis	ter mode (a	any CPU re	gister)
	01 i	ndirect mod	e (disp = 0,	1, IR0, IR1) regis	ter mode (a	any CPU re	gister)
	10 r	register mod	e (any CPL	l register)	indir	ect mode (d	disp = 0, 1,	IR0, IR1)
	11 i	ndirect mod	e (disp = 0,	1, IR0, IR1) indir	ect mode (d	disp = 0, 1,	IR0, IR1)

Type 2

Т	src1 addressing modes	<i>src2</i> addressing modes
00	register mode (any CPU register)	8-bit signed immediate
01	register mode (any CPU register)	indirect mode *+ARn(5-bit unsigned displacement)
10	indirect mode *+ARn(5-bit unsigned displacement)	8-bit signed immediate
11	indirect mode *+ARn1(5-bit unsigned displacement)	indirect mode *+ARn2(5-bit unsigned displacement)

Operation *src1 – src2*

Description The *src2* operand is subtracted from the *src1* operand. The result is not loaded into any register. This allows for nondestructive compares. The *src1* and *src2* operands are assumed to be signed integers. Although this instruction has only two operands, it is designated as a three-operand instruction because operands are specified in the three-operand format.

Status Bits	LUF LV UF N Z V C	Unaffected 1 if an integer overflow occurs, unchanged otherwise 0 1 if a negative result is generated, 0 otherwise 1 if a zero result is generated, 0 otherwise 1 if an integer overflow occurs, 0 otherwise 1 if a borrow occurs, 0 otherwise
Mode Bit	OVM	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	DB cond ARn	, src							
Operands	<i>src:</i> conditional-branch-addressing modes (B) ARn: auxiliary register								
Opcode									
	31	:	24 23		16	15		8 7	0
	0 1 1 0 1 1	В	ARn	0	cond		register o	or displacement	
Word Fields									
			В	3	src a	ddressinç	y modes	_	
			0		regist	ter			
			1		PC re	elative		_	
Operation	If cond is true If cond is true If src is in register src \rightarrow P If src in PC Else, continue	and n reg file), C. relat	ARn ≥ gister-a tive mo	: 0 : addr ode (essing label o	mode(any regist	er in CPU pi <i>ment</i> +PC+1	rimary- \rightarrow PC.
Description	DB <i>cond</i> signif pipeline must auxiliary regis is decremente	ies a be flu ter is ed an	a stand ushed i s greate id a bra	ard f <i>coi</i> er th anch	branch nd is tru nan or e n is per	that exec ie. If the c equal to 0 formed.	cutes in fou ondition is t , the speci	ir cycles becau true and the sp fied auxiliary r	use the ecified egister
	The auxiliary r condition doe	egist s not	ter is tro deper	eate nd oi	d as a 3 n the a	32-bit sigr uxiliary re	ned integer egister deci	Note that the lrement.	branch
	If the <i>src</i> operand is expressed in register-addressing mode, the contents of the specified register are loaded into the PC. If the <i>src</i> operand is expressed in PC-relative addressing mode, the assembler generates a displacement: $displacement = label - (PC of branch instruction + 1)$. This integer is stored as a 16-bit signed integer in the 16 least-significant bits of the branch instruction word. This displacement is added to the PC of the branch instruction plus 1 to generate the new PC.								
	The 'C4x prov (see Section	/ides 11.2 1	20 co for a lis	nditi st of	on coc conditi	les that c ion mnem	an be used ionics, enc	d with this inst oding, and flag	ruction gs).

Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected		
Mode Bit	OVM	operation is not affected by OVM b	it val	ue.
Cycles	4			
Example	DBLT	AR3,R2		
		Before Instruction		After Instruction
	PC [5Fh	PC	9Fh
	A Do [101	VD 2	11h
	ARS	12h	And	
	R2	9Fh	R2	9Fh]
	R2 LUF	9Fh 0	R2 LUF	9Fh
		0	R2 LUF LV	9Fh 0
	R2 [LUF [LV [UF [12h 9Fh 0 0	R2 LUF LV UF	9Fh 0 0
	R2 [LUF [LV [UF [N [12h 9Fh 0 0 0	R2 LUF LV UF N	9Fh 0 0 0 1
	R2 [LUF [LV [UF [N [Z [12h 9Fh 0 0 0 1 1	R2 LUF LV UF N Z	9Fh 0 0 0 1 0
	R2 [LUF [LV [UF [N [Z [V [12h 9Fh 0 0 0 1 1 0	R2 LUF LV UF N Z V	9Fh 0 0 0 0 1 0

DBcondD	Decrement	and Branch	Conditionally	(Delaye	ed)
---------	-----------	------------	---------------	---------	-----

Syntax	DBcondD ARn, src						
Operands	<i>src</i> : conditional-branch-addressing modes (B) ARn: auxiliary register						
Opcode							
	31	24 23	1615	8 7	0		
	011011	ARn 1	cond	register or <i>displacement</i>			
Word Fields							
		В	src addressing	modes			
		0	register				
		1	PC relative				
Operation	$\begin{array}{l} ARn-1 \rightarrow ARn \\ If \ \textit{cond} \ is \ true \ and \\ If \ \textit{src} \ is \ in \ reg \\ register \ file), \\ \textit{src} \rightarrow PC \\ If \ \textit{src} \ is \ n \ PC\text{-rel} \\ Else, \ continue. \end{array}$	ARN ≥ 0: gister addr lative mode	ressing mode (a	any register in CPU prima s) <i>displacement</i> +PC+3→F	ıry- °C.		
Description	DB <i>cond</i> D signifies delayed branch to cycle branch. If the er than or equal to branch is performe affect the <i>cond</i>).	a delayed be fetched condition is zero, the s ed. (The thr	branch that allow before the PC is s true and the spe pecified auxilian ee instructions f	vs the three instructions after t modified. The effect is a sing ecified auxiliary register is gre y register is decremented and ollowing the DB <i>cond</i> D must r	the gle- eat- d a not		
	The auxiliary register is treated as a 32-bit signed integer. None of the three instructions following DB <i>cond</i> D should modify the program flow. Interrupts are disabled for the duration of the DB <i>cond</i> D instruction. Note that the branch condition does not depend on the auxiliary register decrement.						
	If the <i>src</i> operand the specified regist tive addressing, th bel – (PC of branc the branch instruct a delayed branch.	ed in register ad ed into the PC. If er generates a di on + 3). This disp to generate the i	I in register addressing mode, the contents of d into the PC. If the <i>src</i> is expressed in PC-rela- generates a displacement: <i>displacement</i> = la - + 3). This displacement is added to the PC of generate the new PC. Note that bit 21 = 1 for				
	The 'C4x provides (see Section 14.2	20 conditi for a list of	on codes that ca condition mnem	an be used with this instructi ionics, encoding, and flags).	ion		

Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected		
Mode Bit	OVM	operation is not affected by OVM b	it val	ue.
Cycles	1			
Example	DBZD	AR5, \$+110h		
		Before Instruction		After Instruction
	PC [0h	PC	110h
	AR5	67h	AR5	66h
	LUF	0	LUF	0
	LV	0	LV	0
	UF [0	UF	0
	N	0	Ν	0
	z [1	Z	1
	V	0	V	0
	c [0	С	0

Syntax	FIX sr	c, dst						
Operands	src: dst:	general-add register (any	lressing m y register i	iodes (G in CPU p) primary-register file	2)		
Opcode								
	31	24	4 23	161	5	8 7	0	
	000	00101	0 G	dst		src		
Word Fields								
			G	<i>src</i> ad	dressing modes	-		
			00	registe	er (R0 – R11)	_		
			01	direct				
			10	indirec	t			
			11	immec	liate			
						—		
Operation	fix <i>(src</i>)) ightarrow dst						
Description	The floating-point operand <i>src</i> is converted to the nearest integer less than or equal to it in value, and the result is loaded into the <i>dst</i> register. The <i>src</i> operand is assumed to be a floating-point number and the <i>dst</i> operand is assumed to be a signed integer.							
	The ex	kponent field	of the res	ult regis	ter (if it has one) is	not modified.		
	Intege resent the res	r overflow oc ed as a 32-b sult is satura	curs wher bit 2s-comp ted in the	n the floa plement directior	ating-point number integer. In the case n of overflow.	is too large to be r e of integer overfl	ep- low,	
Status Bits	lf ST (flags a registe	SET COND) are modified. ers.	= 0 and t If ST (SET	he desti COND)	nation register is R) = 1, they are modi	0–R11, the condit fied for all destinat	tion tion	
	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 							
	N Z V C	1 if a negati 1 if a zero re 1 if an integ Unaffected	ve result is esult is ge er overflov	s genera nerated, w occurs	ated, 0 otherwise 0 otherwise a, 0 otherwise			
Mode Bit	OVM o	operation is r	not affecte	d by OV	M bit value.			
Cycles	1							

14-90

FIX R1,R2

	Before Instruction										
R1	0A2820 0000h	1.3454e + 3									
R2	0h										
LUF	0										
LV	0										
UF	0										
N	0										
Z	0										
V	0										
С	0										

	After Instruction									
R1	0A2820 0000h	1.3454e + 3								
R2	541h	1345								
LUF	0									
LV	0									
UF	0									
Ν	0									
Z	0									



С 0

Syntax	FI ∥ S ⁻	X src2, de FI src3, de	st1 st2					
Operands	src2: dst1: src3: dst2:	indirect (register (register (indirect (disp = 0, 1 R0 – R7) R0 – R7) disp = 0, 1	, IR0, IR1 , IR0, IR1)			
Opcode								
	31 1 1 0	1010	24 23 dst1	0 0 0	16 <i>src3</i>	15 dst2	8 7 si	0 rc2
Word Fields	None							
Operation	fix(: <i>src</i> .	src2) $ ightarrow$ a $3 ightarrow$ dst2	lst1					
Description	A floating-point-to-integer conversion is performed. All registers are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation be- ing performed in parallel (FIX) writes to the same register, then STI accepts as input the contents of the register before it is modified by FIX.							
	If <i>src2</i> and <i>dst2</i> point to the same location, <i>src2</i> is read before the write to <i>dst2</i> .							
	Intege resent the res	r overflow ed as a 3 sult is satu	occurs wh 2-bit 2s-co urated in th	nen the flo mplemer ne directio	pating-po nt integer on of ove	oint number i r. In the case erflow.	is too large e of integei	to be rep- overflow,
Status Bits	LUF LV UF N Z V C	Unaffecte 1 if an int 0 1 if a neg 1 if a zer 1 if an int Unaffecte	ed leger overf lative resu o result is g leger overf ed	low occu It is gene generate low occu	rs, uncha rated, 0 d d, 0 othe rs, 0 othe	anged other otherwise rwise erwise	wise	
Mode Bit	OVM	operation	is not affe	cted by C	VM bit v	alue.		
Cycles	1							

179

220

220

1.79750e + 02

0

0

С

Example
Example

 $\|$

С

FIX	*++AR4	(1),R1			
STI	R0,*AR	2			
	Before Ins	struction		After Instr	uction
AR4	80 98A2h		AR4	80 98A3h	
R1	0h	66	R1	0B3h	
R0	0DCh	220	R0	0DCh	
AR2	80 983Ch		AR2	80 983Ch	
Data <u>at 8</u>	0 98A3h 733 C000h	1.79750e + 02	Data at 80	0 98A3h 733 C000h	1.79
Data at 8	<u>0 983Ch</u> 0h	2	Data at 8	0 983Ch 0DCh	
LUF 📃	0		LUF 🦳	0	
LV	0		LV 🗌	0	
UF	0		UF 🗌	0	
N	0		N	0	
z 🗌	0		Z 🗌	0	
V [0		V	0	

0

Syntax	FLOAT src, dst	FLOAT src, dst								
Operands	<i>src:</i> general-addressir <i>dst</i> : register (R0 – R11	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (R0 – R11)								
Opcode										
	31 24 23	1615	8 7	0						
	000001011 G	dst	src							

Word Fields

G	src addressing modes
00	register (any register in CPU primary-register file)
01	direct
10	indirect
11	immediate

Operation	float (src) \rightarrow dst					
Description	The integer operand <i>src</i> is converted to the floating-point value equal to it, and the result loaded into the <i>dst</i> register. The <i>src</i> operand is assumed to be a signed integer, and the <i>dst</i> operand is assumed to be a floating-point number.					
Status Bits	 LUF Unaffected LV Unaffected UF 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 0 C Unaffected 					
Mode Bit	OVM operation is not affected by OVM bit value.					
Cycles	1					

Before InstructionAfter InstructionAR280 9800hAR280 9802h80 9802h	
AR2 80 9800h AR2 80 9802h	
R5 034C 2000h 1.27578125e + 01 R5 072E0 0000h 1.74e -	02
Data <u>at 80 9802h</u> Data <u>at 80 9802h</u>	
0AEh 174 0AEh	.74
UF 0 UF 0	
N 0 N 0	
Z 0 Z 0	
V 0 V 0	
C 0 C 0	

FLOAT||**STF** Parallel FLOAT and STF

Syntax	FL ST	LOAT src2 IF src3, de	, dst1 st2						
Operands	src2: dst1: src3: dst2:	indirect (d register (F register (F register (d	lisp = 0, 1, R0 – R7) R0 – R7) disp = 0, 1	, IR0, IR1 , IR0, IR1)				
Opcode									
	31	1011	24 23	0 0 0	16	15 det2	8 7	cro?	0
			USIT	000	SICO	USIZ		5102	
Operation	floa <i>src</i> a	t(src2) \rightarrow 3 \rightarrow dst2	dst1						
Description	An intention of the be one of being cepts	eger-to-floa ginning an the parall performed as input th	ating-point of loaded el operation in parallel e contents	t convers at the endons (STF (FLOAT) s of the re	ion is pe d of the) reads f writes t egister b	erformed. All execute cyc from a regis o the same r efore it is m	l regis le. Th ter an registe odifie	iters are rea is means th id the opera er, then STF d by FLOA ⁻	ad at nat if ation = ac- T.
	If <i>src2</i>	and <i>dst2</i> p	oint to the	same loc	ation, <i>sr</i>	c2 is read be	efore t	he write to a	dst2.
Status Bits	LUF LV UF N Z V C	Unaffecte Unaffecte 0 1 if a nega 1 if a zero 0 Unaffecte	d d ative resul result is (d	t is gener generated	rated, 0 d, 0 othe	otherwise rwise			
Mode Bit	OVM o	operation i	s affected	by OVM	bit value	Э.			
Cycles	1								

Parallel FLOAT and STF FLOAT ||STF

FLOAT *+AR2(IR) STF R7,*AR1	0),R6	
Before Instru	ction	After Instruction
AR2 80 98C5h		AR2 80 98C5h
IR0 8h		IR0 8h
R6 Oh		R6 072E00 0000h 1.740e + 02
R7 034C20 0000h 1.275	578125e + 01	R7 034C20 0000h 1.27578125e + 01
AR1 80 9933h		AR1 80 9933h
Data at 80 98CDh	174	Data at 80 98CDh 0AEh 174
Data at 80 9933h		Data at 80 9933h 034C 2000h 1.27578125e + 01
LUF 0		LUF 0
LV 0		LV 0
UF 0		UF 0
N 0		N 0
Z 0		Ζ 0
V 0		V 0
C 0		C 0

Example

Syntax	FRIEEE sr	c, dst					
Operands	<i>src:</i> direct- or indirect-addressing modes <i>ds</i> t: extended-precision register (R0 – R11)						
Opcode							
	31	24 23		1615	8	7	0
	0 0 0 1 1	1000	G	dst	Sro	с	
Word Fields							
			G	src addressin	ig modes		
		()1	direct			
		1	10	indirect			
Operation	convert src	from IEEE fo	ormat -	→ dst			
Description	The <i>src</i> op complement	erand is conv nt floating-poi	verted int forn	from the IEEI nat.	E floating-poin	t format to the 2	2s-
	The <i>src</i> ope tended pre	erand comes cision registe	from n er as a	nemory. The o single-precisi	converted resu on floating-poi	Ilt goes into an e nt number.	ex-
Status Bits	LUF Una LV Set i UF 0 N Sign Z 1 if r V 1 if c C Una	ffected if overflow, ot of the result result is 0, 0 o overflow, 0 ot ffected	therwis otherwi herwis	e unchanged se e			
Mode Bit	OVM opera	ation is not af	fected	by OVM bit v	alue.		
Cycles	1						
Example	None						

Parallel FRIEEE and STF FRIEEE||STF

Syntax	FF ST	RIEEE sr F sr	rc2, dst1 rc3, dst2				
Operands	src2:indirect mode (disp = 0, 1, IR0, IR1)dst1:register mode (R0 - R7)src3:register mode (R0 - R7)dst2:indirect mode (disp = 0, 1, IR0, IR1)						
Opcode							
	31 1 1 1	1001	24 23 dst1	0 0 0	16 src3	15 8 dst2	7 0 src2
Operation	convert <i>src2</i> from IEEE format \rightarrow <i>dst1</i> in parallel with <i>src3</i> \rightarrow <i>dst2</i>						
Description	The <i>src2</i> operand is converted from the IEEE floating-point format to the 2 complement format. The converted result goes into an extended-precision register <i>dst1</i> as a single-precision floating-point number. A floating-point store is done in parallel. If <i>src2</i> and <i>dst2</i> point to the same location, then <i>src2</i> is read before the write to <i>dst2</i> .						
Status Bits	 LUF Unaffected LV Set if overflow, otherwise unchanged UF 0 N Sign of the result Z 1 if result is 0, 0 otherwise V 1 if overflow, 0 otherwise C Unaffected 						
Mode Bit	OVM operation is not affected by OVM bit value.						
Cycles	1						
Example	None						
IACK Interrupt Acknowledge

Syntax	IACK sr	с								
Operation	<i>src:</i> general-addressing modes (G)									
Opcode										
	31	24	23		1615	8	7	0		
	000	1 1 0 1 1	0 G	0 0 0	0 0	S	rc			
Word Fields										
			G	srca	address	ing modes				
			01 10	direc	oct					
			10	Indir	eci					
Description	At end of A dumm At the en fied . Th edge. Th acknowl Note that	y-read ope nd of the du is instructione IACK sig edge to ext at the IACK	ration at immy re in can be gnal and ternal de signal is	address ad, IAC e used t the add evices. T s extenc	a pointe pointe k is set o gener dress ca he data led with	d by <i>src</i> is perfo to 1 if off-chip rate an external an then be used a read by the pr n multicycle read	rmed with IACK = memory is spe I-interrupt acknow d to signal interru ocessor is unuse ds.	⊧0. ci- wl- upt ∋d.		
Status Bits	LUF U LV U UF U N U Z U V U C U	Inaffected Inaffected Inaffected Inaffected Inaffected Inaffected								
Mode Bit	OVM op	eration is r	ot affect	ted by C	OVM bit	value.				
Cycles	1									

IACK *AR5

	Before Instruction
IACK	1
PC	300h
LUF	0
LV	0
UF	0
N	0
Z	0
V	0
С	0

	After Instruction
IACK	1
PC	301h
LUF	0
LV	0
UF	0
N	0
Z	0
V	0
С	0

Assembly Language Instructions 14-101

Syntax	IDLE
Operands	None
Opcode	
	<u>31 24 23 1615 8 7 0</u>
	0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Word Fields	None
Operation	$1 \rightarrow ST(GIE)$ Next PC \rightarrow PC Idle until interrupt
Description	The global-interrupt enable bit is set, the next PC value is loaded into the PC, and the CPU idles until an unmasked interrupt is received. When the interrupt is received, the contents of the PC are pushed onto the active system stack, and the processor jumps to execute the interrupt service routine.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

Syntax Operands	IDI No	_E2 ne			('C	40	re	visi	on	≥ 5.() ai	nd	' C	44 (on	ly)									
Opcode	31 0	0 0	0 0	1 1	24 0	23 0	0	0 0	0 0	0 0	0	16 0	615 00	0	0	0 0) (8) (7 0	0	0 (0 0	0	0 0	
Word Fields	No	ne																							
Operation	1 - Ne Idle	→ S ⁻ xt P e un	T(GI ℃ → til in	E) ▶ PC terru	pt																				
Description	The IDLE2 instruction performs the same function as IDLE, except that it re- moves the functional clock input from the internal device. This allows for an extremely low-power mode. The PC is incremented once, and the device re- mains in an idle state until one of the external interrupts ($\overline{\text{NMI}}$ or $\overline{\text{IIOFx}}$) is as- serted.																								
	In IDLE2 mode, the 'C4x behaves as follows:																								
		Th	e Cl	PU, p	erip	bhei	rals	s, a	nd ı	nem	ory	y r	etai	n th	nei	r pr	ev	/ioi	us s	sta	tes	•			
		Wł wit	nen t h H	he do 1 higi	evic 1 ar	e is nd ⊦	in 1 3	the Iow	fun	ctior	nal	(n	one	mu	lat	ion) n	noo	de,	the	e clo	clocks stop			
		The 'C4x remains in IDLE2 until one of the external interrupts (NMI IIOFx) is asserted for at least two H1 clock cycles. Then, the clocks st after a delay of one H1 cycle. The clocks can start up in the phase oppose that in which they were stopped (that is, H1 might start high when H3 w high before stopping, and H3 might start high when H1 was high before stopping). However, the H1 and H3 clocks remain 180° out of phase w each other									T or start site was fore with														
		During IDLE2 operation, for one of the external interrupts to be recognized and serviced by the CPU, it must be asserted for at least two H1 cycles For the processor to recognize only one interrupt when it restarts opera tion, the interrupt pin must be configured for edge-triggered mode or as serted for less than three cycles in level-triggered mode.									zed les. era- as-														
	❑ When the 'C4x is in the emulation mode, the H1 and H3 clocks continue to run normally, and the CPU operates as if an IDLE instruction had been executed. The clocks continue to run for correct operation of the emulator.										nue een itor.														
		An CF inte the	y ex PU to errul e IDL	terna o reco ot is r _E2 ii	il in ogni ecc nstr	terri ize i ogni: ucti	upt tha zec on	t pir at in d ar is i	n ca terr nd e not	n wa upt, xecı exec	ike the iteo ute	e up e ir d b ed	o th nter oy th unt	e de rupi ne C il at	ev t n CP fte	ice nusi U, t r a	frc t a he ret	om Isc in tur	IDI be stru n is	LE: ei ucti ; e:	2; b nab ion xec	ut f led foll ute	for I. II ow	the f an /ing	

Status Bits	LUF	Unaffected
	LV	Unaffected
	UF	Unaffected
	Ν	Unaffected
	Z	Unaffected
	V	Unaffected
	С	Unaffected
Mode Bit	ΟνΜ	operation is not affected by OVM bit value
Cycles	1	
Example	None	

Syntax	LAJ src										
Operands	src: in PC-relative mode										
Opcode											
	31	24	23	1615		8 7	0				
	0 1 1 0	0 0 1 1			displacement						
Word Fields	None										
Operation	PC of LAJ + 4 \rightarrow extended-precision register R11 displacement + 3 + PC of LAJ \rightarrow PC										
Description	LAJ performs a single-cycle delayed subroutine call that allows the three in- structions following the LAJ instruction to be performed before branching. The return address (address of the LAJ instruction + 4) is placed in extended-preci- sion register R11. The assembler generates a displacement: <i>displacement</i> = <i>src</i> – (PC of branch instruction + 1). This displacement is stored as a 24-bit signed integer in the 24 least significant bits of the branch instruction. This dis- placement is added to the PC of the branch instruction plus 1 to generate the new PC. See Section 6.6 on page 6-19 for details.										
	None of the three instructions following the LAJ instruction should modify R11 or the program flow. Interrupts are disabled for the duration of the LAJ instruction.										
Status Bits	LUF U LV U UF U N U Z U V U C U	Inaffected Inaffected Inaffected Inaffected Inaffected Inaffected									
Mode Bit	OVM op	eration is n	ot affected b	y OVM bi	t value.						
Cycles	1										
Example	None										

Syntax Operands Opcode	LAJcon src:	<i>nd src</i> conditional	-branch ac	dressing mode	S	8 7	0
	0 1 1	100 B	0 0 0 1	cond	register o	or displacement	Ť
Word Fields							
			В	<i>src</i> addressir	ng modes	_	
			0	register			
			1	PC relative		_	
Operation	If (conc If (si P si If (si P di Else, ci	d is true) rc is a regis C of LAJcc rc \rightarrow PC rc is in PC- C of LAJcc isplacement isplacement	ster) $pnd + 4 \rightarrow r$ $pnd + 4 \rightarrow r$ $pnd + 4 \rightarrow r$ pnt = src - (F) pnt + PC of t	extended-precised ode) extended-precise C of LAJ + 3) he LAJ + 3 \rightarrow F	sion register sion register PC	r R11 r R11	
Description	LAJ <i>cor</i> lows th before the LA dress b	nd performs e three ins branching, J instruction pranched to	s a conditions for without aft n + 4) is plat o is formed	onal single-cycl Ilowing the LA fecting the <i>conc</i> aced in extende by either regist	e delayed s I <i>cond</i> instruct d. The return d-precision ter mode or	ubroutine call t ction to be perf address (add register R11. T PC-relative mo	that al- formed ress of The ad- ode.
	None of modify the LA.	of the three R11 or th J <i>cond</i> instr	e instruction e program uction.	ons following t n flow. Interrupt	he LAJ <i>con</i> is are disabl	d instruction s led for the dura	should ation of
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected					
Mode Bit	ОVМ о	peration is	not affecte	ed by OVM bit v	alue.		
Cycles	1						
Example	None						

Syntax	LATcond N									
Operands	N immediate mode – trap number ($0 \le N \le 511$)									
Opcode										
	31 24 23 1615 8 7 0 0 1 1 0 0 0 0 0 0 0 N									
Word Fields	None									
Operation	$\begin{array}{l} \mbox{If (cond is true)} \\ ST(GIE) \rightarrow ST(PGIE) \\ ST(CF) \rightarrow ST(PCF) \\ 0 \rightarrow ST(GIE) \\ 1 \rightarrow ST(CF) \\ \mbox{PC of } LA \mbox{ cond } + 4 \rightarrow extended\mbox{-precision register } R11 \mbox{ trap vector } N \rightarrow PC \\ \mbox{Else, continue.} \end{array}$									
Description	The LAT <i>cond</i> instruction performs a conditional delayed single-cycle trap. If the condition is true, ST bits GIE and CF are saved in PGIE and PCF in the status register. Then all interrupts are disabled ($0 \rightarrow$ GIE), and the cache is frozen ($1 \rightarrow$ CF). The contents of the PC of the LAT <i>cond</i> + 4 are placed in R11, and the PC is loaded with the contents of the specified trap vector (N). If the condition is not true, then continue normal operation. If traps are to be nested, you may need to save the status register before executing LAT <i>cond</i> .									
	The three instructions following LAT <i>cond</i> are fetched and executed, but they do not affect the <i>cond</i> . They should not modify the program flow or directly modify the status register. Interrupts are disabled for the duration of the LAT <i>cond</i> N instruction.									
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected									
Mode Bit	OVM operation is not affected by OVM bit value.									
Cycles	1									
Example	None									

Syntax	LBb	src, dst								
Operands	<i>src</i> : register, direct, 16-bit immediate, or indirect-addressing modes <i>dst</i> : register mode (any register in CPU primary-register file)									
Opcode										
	31	24 23 1615		8 7 0						
	1 0	1 1 0 0 0 B G dst		SrC						
Word Fields										
	G	src addressing modes	В	src byte						
	00	register mode	00	byte 0 LS byte						
	01	direct mode	01	byte 1						
	10	indirect mode	10	byte 2						
	11	immediate mode (16 bits)	11	byte 3 MS byte						
Operation	Sign-	extended byte $(3, 2, 1, 0)$ of erc \rightarrow	det							
operation	Olgh	$= (0, 2, 1, 0) \text{is } c \rightarrow c$	<i>u31</i>							
	b = b	yte to load (3, 2, 1, 0)								
	3	2 1 0 = b (byte designate	or 3 -	- 0)						
Description	The s the e mode instru ue of	specified byte of the <i>src</i> operand is ight LSBs of the <i>dst</i> register. The <i>s</i> is specified and byte 2 (B =10) or iction performs sign extension of the 00h or FFh is stored into the eight	sign src b byte 16- LSB	-extended and right-shifted into yte is signed. When immediate 3 (B = 10) is selected, the LBb bit value. Consequently, the val- s of the <i>dst</i> register.						
Status Bits	If ST flags regist	(SET COND) = 0 and the destination are modified. If ST (SET COND) = 1 ters.	on re , the	gister is R0 – R11, the condition by are modified for all destination						
	LUF LV UF N Z V C	Unaffected Unaffected 0 1 if a negative result is generated 1 if a zero result is generated, 0 o 0 Unaffected	, 0 o ther	therwise wise						
Mode Bit	ονμ	operation is not affected by OVM b	oit va	lue.						
Cycles	1									
14-108										

Example	LB2	R1,	R2	;	sign	extended	byte	2	of	R1	\rightarrow R2
		Be	efore In	struc	ction				After	Inst	ruction
	R1	00AB (0000h			R	1	00A	B 000)0h	
	R2	0000	0000h			R	2 F	FFF	FFA	Bh	

Syntax	LBU	o src, dst				
Operands	src: dst:	register, direct, 16- register mode (any	oit immedia register in	ate, or in CPU pri	direct-addressing modes imary-register file)	
Opcode						
	31	24 23	161	5	870	
	10	1 1 0 0 1 B G	dst		STC	
Word Fields						
	G	<i>src</i> addressing mod	es	В	<i>src</i> byte	
	00	register mode		00	byte 0 LS byte	
	01	direct mode		01	byte 1	
	10	indirect mode		10	byte 2	
	11	immediate mode (16	oits)	11	byte 3 MS byte	
Operation	Byte	$(3, 2, 1, 0) \text{ of } src \to c$	lst			
	b = t	oyte to load (3, 2, 1, 0))			
	3	2 1 0 =) (byte desię	gnator 3 -	- 0)	
Description	The specified byte of the <i>src</i> operand is right-shifted, without sign extension, into the eight LSBs of the <i>dst</i> register. The <i>src</i> byte is unsigned. When immediate mode is specified and byte 2 ($B = 10$) or byte 3 ($B = 10$) is selected, the LBUb instruction performs sign extension of the 16-bit value. Consequently, the value of 00b or EEb is stored into the eight LSBs of the <i>dst</i> register.					
Status Bits	If ST flags regist	(SET COND) = 0 and are modified. If ST (S ters.	I the destin ET COND)	nation reg) = 1, the	gister is R0 – R11, the condition y are modified for all destination	
	LUF LV UF N Z V C	Unaffected Unaffected 0 0 1 if a zero result is 0 Unaffected	generated,	0 other	wise	
Mode Bit	ΟνΜ	operation is not affe	cted by OV	/M bit va	lue.	
Cycles	1					
Example	LBU2 R1 [R2 [R1, R2 Before Instruction 00AB 0000h 0000 0000h	n	R1 R2	After Instruction	

Syntax	LDA src, dst						
Operands	<i>src</i> : general-addre <i>dst</i> : register mode	essing m e (addres	odes s registers only)				
Opcode							
	31 24 2	23	1615	8 7 0			
	0 0 0 1 1 1 1 0 1	G	dst	SIC			
Word Fields							
	-	G	src addressing modes	_			
	-	00	register (any register in CPU primary-register file)				
		01	direct				
		10	indirect				
	_	11	immediate				
Operation	src ightarrow dst						
Description	The <i>src</i> operand is loaded into the dst register. The <i>dst</i> register can be any of the address registers: AR0 – AR7, IR0, IR1, DP, BK, or SP. The load is complete by the end of the read phase of the pipeline. As a result, LDA is one cycle faster than LDI for loading these registers. (All operands are treated as signed integers.)						
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected						
Mode Bit	OVM operation is no	ot affecte	d by OVM bit value.				
Cycles	1						
Example	None						

Syntax	LDE src, dst			
Operands	<i>src:</i> general-addr <i>dst</i> : register (R0 -	essing mo - R11)	des (G)	
Opcode				
	31 24	23	1615	8 7 0
	00000110	I G	dst	SIC
Word Fields				
	-	G	src addressing modes	-
	-	00	register (R0 – R11)	_
		01	direct	
		10	indirect	
	_	11	immediate	_
Operation	$\textit{src}(exp) \rightarrow \textit{dst}(exp)$			
Description	The exponent field of dst register. No moothe value of the exponent as determined by the the dst register is setting-point numbers. format.	of the <i>src</i> of lification of onent loade e precisior at to 0. The Immediate	operand is loaded into the f the <i>dst</i> register mantissa ed is the reserved value of n of the <i>src</i> operand. Then e <i>src</i> and <i>dst</i> operands are e values are evaluated in th	exponent field of the a field is made unless the exponent for zero , the mantissa field of e assumed to be floa- ne short floating-point
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected			
Mode Bit	OVM operation is no	ot affected	by OVM bit value.	
Cycles	1			

LDE R0,R5

	Before Ir	struction
R0	020005 6F30h	4.00066337e + 00
R5	0A056F E332h	1.06749648e + 03
LUF	0	
LV	0	
UF	0	
N	0	
Z	0	
V	0	
C	0	

After Instruction

R0	020005 6F30h	4.00066337e + 00
R5	02056F E332h	4.16990814e + 00
LUF	0	
LV	0	
UF	0	
Ν	0	
Ζ	0	

0 0

Ν Ζ ٧

С

Assembly Language Instructions 14-113

Syntax	LDEP src, dst
Operands	src: expansion register file register (IVTP or TVTP)dst: register mode (any register in CPU primary register file)
Opcode	
	31 24 23 1615 8 7 0 0 1 1 1 0 1 1 0 0 0 0 dst 0 0 0 0 0 0 0 0 0 0 0 src
Word Fields	None
Operation	$src \rightarrow dst$
Description	The LDEP instruction loads a CPU register with the contents of the IVTP register (interrupt-trap table pointer) or the TVTP register. These registers are described in Section 3.2.
	The <i>src</i> operand register from the expansion-register file is loaded into the <i>dst</i> register in the primary register file. The <i>dst</i> register content is assumed to be an integer.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedXUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

Syntax	LDF :	src, de	st					
Operands	src: dst:	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (R0 – R11)						
Opcode								
	31		2	24 23		1615		8 7 0
	000	0 0 0	1 1	10	G	dst		src
Word Fields								
				_	G	<i>src</i> add	ressing modes	_
					00	register	(R0 – R11)	_
					01	direct		
					10	indirect		
					11	immedia	ate	
								_
Operation	src \rightarrow	dst						
Description	The <i>s</i> assur	<i>rc</i> ope ned to	erand i be flo	is loa batin	aded i g-poir	nto the <i>dst</i> nt numbers	register. The <i>dst</i>	and <i>src</i> operands are
Status Bits	LUF LV UF N Z V C	 Unaffected Unaffected 0 1 if a negative result is loaded, 0 otherwise 1 if a zero result is loaded, 0 otherwise 0 Unaffected 						
Mode Bit	ονΜ	opera	tion is	s not	affect	ed by OVM	I bit value.	
Cycles	1							

LDF Load Floating-Point Value

LDF @9800h,R2		
Before II	nstruction	After Instruction
DP 80h		DP 80h
R2 0h		R2 010C52 A000h 2.19254303e + 00
Data <u>at 80 9800h</u>		Data <u>at 80 9800h</u>
10C5 2A00h	2.19254303e + 00	10C5 2A00h 2.19254303e + 00
LUF 0		LUF 0
LV 0		LV 0
UF 0		UF 0
N 0		N 0
Z 0		Ζ 0
V 0		V 0
C 0		C 0

Example

Syntax	LDFcond src, dst					
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (R0 – R11)					
Opcode						
	31 24 23	1	615	8 7	0	
	0 1 0 0 <i>cond</i> G	dst		SIC		
Word Fields						
	—	G	src addressing mo	des		
		00	register (R0 – R11)			
		01	direct			
		10	indirect			
		11	immediate			
Operation Description	If <i>cond</i> is true: $src \rightarrow dst.$ Else: dst is unchanged. If the condition is true wise, the dst register	e, the <i>sr</i> e	c operand is loaded anged. The <i>dst</i> and	into the <i>dst</i> register. O <i>src</i> operands are assu	ther- imed	
	to be floating-point numbers. The 'C4x provides 20 condition codes that can be used with this instruction (see Section 14.2 on page 14-12 for a list of condition mnemonics, encoding, and flags). Note that an LDFU (load floating-point unconditionally) instruction is useful for loading R0 – R11 without affecting condition flags.					
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected					
Mode Bit	OVM operation is not	affected	d by OVM bit value.			
Cycles	1					

LDFcond Load Floating-Point Value Conditionally

Example







14-118

Syntax	LDFI	src, dst							
Operands	<i>src</i> : general-addressing modes (G) <i>dst:</i> register (R0 – R11)								
Opcode									
	31		24 23			1615	8 7	7	0
	0 0 0	0011	1 1	G	dst		src		
Word Fields									
				(9	<i>src</i> address modes	sing		
				C	1	direct			
				1	0	indirect			
Operation	Signa <i>src</i> →	l interlocke <i>dst</i>	d ope	eratio	n.				
Description	The <i>s</i> naled ting-p tion 9	rc operand over LOCK oint numbe .7 (page 9-3	is loa or LL rs. Oi 39) fo	ded in OCK nly di or a d	nto the . The s rect an etailec	<i>dst</i> register. A src and <i>dst</i> ope d indirect mod l description.	n interlocke rands are as les are allov	d operation is s ssumed to be flo ved. Refer to So	sig- oa- ec-
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected 0 1 if a nega 1 if a zero 0 Unaffected	d d ative i resu d	resulf It is g	is ger enerat	nerated, 0 othe red, 0 otherwis	erwise se		
Mode Bit	OVM	operation is	s not	affec	ted by	OVM bit value	Э.		
Cycles	1								

LDFI *+AR2,R7

Before Ins	struction		After Instruction			
AR2 8098F1h		AR2	8098F1h			
R7 0h		R7	0584C0 0000h	-6.28125e + 01		
Data at 80 98F2h 584 C000h	-6.28125e + 01	Data	at 80 98F2h 584 C000h	-6.28125e + 01		
LUF 0		LUF	0			
LV 0		LV	0			
UF 0		UF	0			
N 0		Ν	0			
Z 0		Z	0			
V 0		V	0			
C 0		С	0			

Syntax	LD LD	₽F src2, ds ₽F src1, ds	t2 t1					
Operands	src1: dst1: src2: dst2:	indirect (di register (R indirect (di register (R	sp = 0, 0 — R7 sp = 0, 0 — R7	1, IR0, I) 1, IR0, I)	R1) R1)			
Opcode								
	31	0 0 1 0	24 23 det2	det1	16	15 src1	8 7	0
		0010	<i>US12</i>	usii	000	5101		5102
Word Fields	None							
Operation	src2 src1	$2 \rightarrow dst2$ $1 \rightarrow dst1$						
Description	Two flo registe	ating-poin r, the asse	t loads a mbler is:	are perfo sues a v	ormed in varning.	parallel. If th The result is	that of LD	ad the same F <i>src2, dst2.</i>
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected	5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7					
Mode Bit	OVM o	peration is	s not affe	ected by	OVM bi	t value.		
Cycles	1							

LDF||**LDF** *Parallel LDF and LDF*

Example

LDF * AR1 (I	R0),R7	
Before Instru	, R3 uction	After Instruction
AR1 80 985Fh		AR1 80 9857h
IR0 8h		IR0 8h
R7 Oh		R7 070C80 0000h 1.4050e + 02
AR7 80 988Ah		AR7 80 988Bh
R3 Oh		R3 057B40 0000h 6.281250e + 01
Data <u>at 80 9857h</u> 70C 8000h	1.4050e + 02	Data <u>at 80 9857h</u> 70C 8000h 1.4050e + 02
Data at 80 988Ah 57B 4000h 6	.281250e + 01	Data at 80 988Ah 57B 4000h 6.281250e + 01
LUF 0		LUF 0
LV 0		LV 0
UF 0		UF 0
N 0		N 0
Z 0		Ζ 0
V 0		V 0
C 0		C 0

Syntax	LDF s ∥ STF s	src2, dst1 src3, dst2				
Operands	<i>src2</i> : indi <i>dst1</i> : reg <i>src3:</i> reg <i>dst2</i> : indi	irect (disp = 0, 1 ister (R0 – R7) ister (R0 – R7) irect (disp = 0, 1	, IR0, IR1 , IR0, IR1)		
Opcode						
	31	24 23	0 0 0	16 cro2	15 8	7 0
			0 0 0	5165	USIZ	5102
Word Fields	None					
Operation	$src2 \rightarrow$ $src3 \rightarrow$	dst1 dst2				
Description	A floating-	point load and a	floating-	point sto	ore are perform	ed in parallel.
	If <i>src2</i> and	I <i>dst2</i> point to the	same loc	ation, <i>si</i>	c2 is read befor	re the write to <i>dst2.</i>
Status Bits	LUF Una LV Una UF Una N Una Z Una V Una C Una	affected affected affected affected affected affected affected				
Mode Bit	OVM oper	ration is not affe	cted by C	VM bit v	value.	
Cycles	1					

LDF||STF Parallel LDF and STF

Example

LDF *AR2(STF R3,*AR4	(1),R1 ++(IR1)		
Before In	struction	After Instru	uction
AR2 80 98E6h		AR2 80 98E6h	
R1 0h		R1 070C80 0000h	1.4050e + 02
R3 057B40 0000h	6.28125e + 01	R3 057B40 0000h	6.28125e + 01
AR4 80 9900h		AR4 80 9910h	
IR1 10h		IR1 10h	
Data <u>at 80 98E7h</u> 70C 8000h	1.4050e + 02	Data at 80 98E7h 70C 8000h	1.4050e + 02
Data at 80 9900h		Data at 80 9900h 57B 4000h	6.28125e + 01
LUF 0		LUF 0	
LV 0		LV 0	
UF 0		UF 0	
N 0		N 0	
Z 0		Z 0	
V 0		V 0	
C 0		C 0	

14-124

Syntax	LDHI src, dst			
Operands	<i>src</i> : 16-bit unsigned immediate <i>dst</i> : register mode			
Opcode				
	<u>31 24 23 1615 8 7 0</u>			
	0 0 0 1 1 1 1 1 1 1 1 <i>dst src</i> (immediate value)			
Word Fields	None			
Operation	$src \rightarrow 16$ MSBs of dst			
Description	The 16-bit unsigned <i>src</i> immediate value is loaded into the 16 MSBs of the <i>dst</i> register, and 0 is loaded into the 16 LSBs of the <i>dst</i> register. The <i>dst</i> register is assumed to be an integer.			
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected			
Mode Bit	OVM operation is not affected by OVM bit value.			
Cycles	1			
Example	LDHI 44h, R2			
	Before Instruction After Instruction			

Syntax	LDI si	c, dst					
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)						
Opcode							
	31	1	24 23	;	1615		8 7 0
	000	0 1 0 0	0 0	G	dst		SrC
Word Fields	None						
				G	<i>src</i> addre	essing modes	-
				00	register (a CPU prim	any register in ary-register file)	_
				01	direct		
				10	indirect		
				11	immediate	e	_
Operation	src ightarrow	dst					
Description	The <i>si</i> assum	rc operand red to be s	l is loa signec	aded int d intege	o the <i>dst</i> re rs.	egister. The <i>dst</i> a	and <i>src</i> operands are
Status Bits	LUF LV UF N Z V C	Unaffecte Unaffecte 0 1 if a neg 1 if a zero 0 Unaffecte	ed ed ative o resu ed	result is It is ger	generated	d, 0 otherwise otherwise	
Mode Bit	OVM	operation	is not	affected	d by OVM	bit value.	
Cycles	1						

Load Integer LDI

38

38

Example

LDI *-AR1(IR0),R5

	· · · · · ·
	Before Instruction
AR1	2Ch
IR0	5h
R5	3C5h
Data	at 27h
	26h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

965

38

С

		After Instruction
AR1		2Ch
IR0		5h
R5		26h
Data	at 27h	
		26h
LUF		0
LV		0
UF		0
Ν		0
Z		0
V		0

0

Assembly Language Instructions 14-127

LDIcond Load Integer Conditionally

Syntax	LDIcol	nd src, dst				
Operands	src: dst:	general ade register (ar	dressing m ny register	iodes (G) in CPU primary i	register file)	
Opcode						
	31	1 cond	24 23	1615 det	<u> </u>	0
	0 1 0		G	usi	SIC	
Word Fields						
			G	<i>src</i> addressing	g modes	
			00	register (any re CPU primary re	gister in egister file)	
			01	direct		
			10	indirect		
			11	immediate		
Operation	lf <i>cond</i> src - Else: d	′ is true: → <i>dst,</i> <i>lst</i> is uncha	nged.			
Description	If the c wise, tl to be s	ondition is he <i>dst</i> regis igned integ	true, the <i>si</i> ster is unch ers.	r <i>c</i> operand is loa nanged. The <i>dst</i>	ded into the <i>dst</i> registe and <i>src</i> operands are a	r. Other- assumed
	LDP (a any oth	n alternate ner register	form of LD with the 10	0IU) loads the da 6 MSBs of a relo	ta-page pointer registe catable address.	r (DP) or
	The 'C (see Se that a le lected of affects	4x provides ection 14.2 oad integer CPU registe	s 20 condit for a list of (unconditio er without a	ion codes that c condition mnemo nally (LDIU) insti ffecting the cond	an be used with this ir onics, encoding, and fla ruction is useful for loac ition flags that the LDI ir	struction gs). Note ling a se- struction
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected				
Mode Bit	OVM o	peration is	not affecte	ed by OVM bit va	llue.	
Cycles	1					
14-128						

Exam	ple
EAGIN	

LDIZ R4,R6

	Before Instruction
R4	027Ch
R6	0FE2h
LUF	0
LV	0
UF	0
N	0
Z	0
V	0
С	0

636

4066

R4

R6

LV

UF

Ν

Ζ

٧ С



Assembly Language Instructions 14-129

LDII Load Integer, Interlocked

Syntax	LDII si	rc, dst			
Operands	src: dst:	general-addr register (any	essing mo register in	des (G) CPU primary-register	file)
Opcode					
	31 0 0 0	0 1 0 0 0	23 1 G	1615 dst	8 7 0 src
Word Fields					
		-	G	<i>src</i> addressing modes	,,,,,,,
			01	direct	
		-	10	indirect	
Operation Description	Signal $src \rightarrow$ The src naled signed	interlocked o <i>dst</i> c operand is lo over LOCK of integers, Not	peration. Daded into r LLOCK. re that only	the <i>dst</i> register. An inte The <i>src</i> and <i>dst</i> opera the direct and indirect	rlocked operation is sig- nds are assumed to be modes are allowed. Re-
	fer to S	Section 9.7 or	n page 9-3	9 for a detailed descrip	ition.
Status Bits	If ST (S flags a registe	SET COND) = re modified. If ers.	= 0 and the ST (SET (e destination register is COND) = 1, they are mo	R0 – R11, the condition odified for all destination
	LUF LV UF N Z V C	Unaffected Unaffected 0 1 if a negativ 1 if a zero res 0 Unaffected	e result is sult is gen	generated, 0 otherwise erated, 0 otherwise	3
Mode Bit	OVM o	peration is no	ot affected	by OVM bit value.	
Cycles	1				

LDII @985Fh,R3

	1 0000111/100
	Before Instruction
DP	80
R3	Oh
Data	at 80 985Fh
	0DCh
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

	After Instruction					
DP	80					
R3	0DCh					
Data at 80 98F5h						
	0DCh					
LUF	0					
LV	0					
UF	0					
Ν	0					
Z	0					
V	0					
С	0					

Assembly Language Instructions 14-131

Syntax	LDI s	erc2, dst. erc1, dst	2 1						
Operands	src1:indirect (disp = 0, 1, IR0, IR1) $dst1:$ register (R0 - R7) $src2:$ indirect (disp = 0, 1, IR0, IR1) $dst2:$ register (R0 - R7)								
Opcode									
	31		24 23			161	15	8 7	0
	1 1 0 0	0 1 1	dst2	dst1	0	00	src1	sre	c2
Word Fields	None								
Operation	$src2 \rightarrow dst2$ $src1 \rightarrow dst1$								
Description	Two integer loads are performed in parallel. A warning is issued by the assembler if the LDIs load the same register. The result is that of LDI <i>src2, dst2.</i>								
Status Bits	LUF Ur LV Ur UF Ur N Ur Z Ur V Ur C Ur	affected affected affected affected affected affected affected	1 1 1 1 1 1 1 1 1 1						
Mode Bit	OVM ope	ration is	not affe	cted by	ov	M bit v	value.		
Cycles	1								

I	DI *-AR1(1),R7						
I	DI *AR7++(IR0),R1						
Before Instruction							
AR1	80 9826h						
R7	0h						
AR7	80 98C8h						
IR0	10h						
R1	0h						
Data	at 80 9825h						
	0FAh	250					
Data	at 80 98C8h						
	2EEh	750					
LUF	0						
LV	0						
UF	0						
Ν	0						
Z	0						
V	0						
С	0						

After Instruction AR1 [80 9826h R7 0FAh 250 AR7 80 98D8h IR0 10h R1 02EEh 750 Data at 80 9825h 0FAh 250 Data at 80 98C8h 2EEh 750 LUF 0 LV 0 UF 0 0 Ν 0 Ζ 0 ۷ С 0

Syntax	LI S ⁻	DI src2, ds TI src3, ds	t1 t2				
Operands	src2:indirect (disp = 0, 1, IR0, IR1 $dst1:$ register (R0 - R7) $src3:$ register (R0 - R7) $dst2:$ indirect (disp = 0, 1, IR0, IR1)						
Opcode							
	31		24 23		16	15 8	7 0
	1 1 0	1 1 0 1	dst1	000	src3	dst2	src2
Word Fields	None						
Operation	$src2 \rightarrow dst1$ $src3 \rightarrow dst2$						
Description	An integer load and an integer store are performed in parallel. If <i>src2</i> and <i>dst2</i> point to the same location, <i>src2</i> is read before the write to <i>dst2</i> .						
Status Bits	LUF LV UF N Z V C	Unaffecte Unaffecte Unaffecte Unaffecte Unaffecte Unaffecte	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2				
Mode Bit	OVM	operation i	s not affec	ted by O	VM bit v	alue.	
Cycles	1						

LDI *-AR1(1),R2							
STI R7,*AR5++(I	R0)						
Before Instruction							
AR1 80 98E7h		AR					
R2 0h		R2					
R7 35h	53	R7					
AR5 80 982Ch		AR					
IR0 8h		IRC					
Data <u>at 80 98E6h</u>	Data at 80 98E6h						
0DCh	220						
Data at 80 982Ch		Da					
0n							
LUF 0		LU					
LV 0		LV					
UF 0		UF					
N 0		Ν					
Ζ 0		Z					
V 0		V					
C 0		С					

After Instruction 80 98E7h **1**1 0DCh 220 35h 53 <u>۱</u>5 آ 80 9834h 0 | 8h ata at 80 98E6h DDCh 220 ta <u>at 80 982Ch</u> 35h 53 0 IF [0 0 0 0 0 0

Assembly Language Instructions 14-135
Syntax	LDM src, dst					
Operands	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (R0 – R11)					
Opcode						
	31 24 2	23	1615	870		
	0 0 0 0 1 0 0 1 0) G	dst	STC		
Word Fields						
	-	G	src addressing modes	_		
	-	00	register (R0 – R11)	_		
		01	direct			
		10	indirect			
	_	11	immediate			
	_					
Operation	src (man) $ ightarrow dst$ (ma	an)				
Description	The mantissa field c dst register. The dst are assumed to be f used, bits $15 - 12$ of the source is in the n	of the <i>src</i> of exponent floating-po the instru nemory, th	operand is loaded into the field is not modified. The bint numbers. If immediate action word are forced to 0 he 32-bit data are loaded ir	e mantissa field of the src and dst operands e addressing mode is by the assembler. If not the mantissa field.		
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected					
Mode Bit	OVM operation is no	ot affected	l by OVM bit value.			
Cycles	1					

Example	Exam	ple
---------	------	-----

LDM 156.75,R2

	•
	Before Instruction
R2	0h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

 $(156.75 = 07 \ 1CC0 \ 0000h)$

After Instruction R2 00 1CC0 0000h 1.22460938e + 00 LUF [0 0 LV UF 0 Ν 0 0 Ζ

- 0 ٧ С 0

Assembly Language Instructions 14-137

Syntax	LDP s	src[,DP]						
Operands	 src: 16 MSBs of the absolute 32-bit source address (src). dst: optional (data-page pointer understood if ",DP" left out of operand) 							
Opcode								
	31 0 1 0	2 1 0 0 0 0	4 23 0 1 1	1 0	16 0 0 0	615	8 7 0 src	
Word Fields	None							
Operation	src $ ightarrow$	Data-page p	pointer					
Description	This p <i>is alwa</i> of the to mal	oseudo-op is <i>ays</i> in the imr <i>src</i> absolute ke the 32 bits	an alterna nediate ao 32-bit val s) are loao	ate forr ddress ue (no ded int	n of the ing mod te that o the 1	e LD de (t an <i>s</i> 6 LS	IU instruction, except that LDP bits $22 - 21 = 11_2$). The 16 MSBs <i>rc</i> less than 32 bits is zero filled BBs of the data-page pointer.	
	The 1 page of words	6 LSBs of th of data being long. Bits 31	e pointer addresse –16 of the	are us ed. The e point	ed in d ere is a er are r	irect tota esei	addressing as a pointer to the l of 64K pages, each page 64K rved and should be kept to zero.	
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected						
Mode Bit	OVM	operation is	not affecte	ed by C	DVM bi	t val	ue.	
Cycles	1							
Example	LDP (or LDP (8809900h, 8809900h	DP					
	DP [6465h	Instruction			DP	After Instruction	
		01001	-			51	src, zeros extended	
		0	ן ר			LUF		
		0	ם ה					
		0	1			N		
	z [0]			z	0	
	v	0]			v	0	
	с[0				С	0	

Syntax	LDPE src, dst							
Operands	<i>src</i> : register mode (any register in CPU primary-register file) <i>dst</i> : expansion-register file register (IVTP or TVTP)							
Opcode								
	<u>31 24 23 1615 8 7 0</u>							
	0 1 1 1 0 1 1 0 1 0 0 dst 0 0 0 0 0 0 0 0 0 0 0 src							
Word Fields	None							
Operation	$src \rightarrow dst$							
Description	This is a means to load the interrupt vector table pointer (IVTP) register or trap- vector table pointer (TVTP) register. These registers are described in Section 3.2 on page 3-17.							
	The <i>src</i> operand register from the primary-register file is loaded into the <i>dst</i> register in the expansion register file. The <i>dst</i> operand is assumed to be an integer.							
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected							
Mode Bit	OVM operation is not affected by OVM bit value.							
Cycles	1							
Example	LDPE AR0, TVTP ; set trap-vector pointer							

Syntax	LDPK src				
Operands	<i>src</i> : 16-bit unsigned immediate				
Opcode					
	31	24 23	1615	8 7	0
	00011	1 1 1 0 1 1 1 0	0 0 0 0	SIC	
Word Fields	None				
Operation	$\textit{src} \rightarrow \text{DP}$				
Description	The 16-bit unsigned immediate value is loaded into the DP register. This oper- ation is completed by the end of the decode phase of the LDPK instruction; thus, the value loaded is ready for the next instruction for immediate address- ing. Use caution when using the DP register in the instruction that precedes the LDPK. For example:				
	PUSH DP LDPK new	_value			
	pushes the I	OP new value inte	o the stack instead o	of saving the old DP	value.
Status Bits	LUF Unaff LV Unaff UF Unaff N Unaff Z Unaff V Unaff C Unaff	ected ected ected ected ected ected ected			
Mode Bit	OVM operat	ion is not affected	d by OVM bit value.		
Cycles	1				

Syntax	LHw src, dst							
Operands	<i>src</i> : register, dir <i>dst</i> : register mo	<i>src</i> : register, direct, 16-bit immediate, or indirect-addressing modes <i>dst</i> : register mode (any register in CPU primary-register file)						
Opcode								
	31 2	24 23		1615	8 7 0			
	1 0 1 1 1 0 1 0) H	G	dst	src			
Word Fields								
			G	src addressing modes	_			
			00	register (any register in CPU primary-register file)				
			01	direct				
			10	indirect				
			11	immediate				
			Н	src half-word				
			0	half-word 0 (LS half-word)	_			
			1	half-word 1 (MS half-word))			
Operation	Sign-extended ha	lf-wo	rd (0,	1) of $src \rightarrow dst$				
	w = half-word to	load	(0, 1)					
	1 0)] = w	designator				
Description	The specified half into the 16 LSBs o ate mode is specif performs sign exte the corresponding	f-word of the fied a ensio half-	d of th <i>dst</i> re nd a h on of t word	ne <i>src</i> operand is sign-exter gister. The <i>src</i> half-word is s half-word 1 (H = 1) is selecte he 16-bit value into a 32-bit value (0000h or FFFFh) is s	nded and right-shifted signed. When immedi- ed, the LHw instruction value. Consequently, tored into the 16 LSBs			

of the dst register.

Status Bits	lf ST flags regis	If ST (SET COND) = 0 and the destination register is $R0-R11$, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.							
	LUF LV UF N Z V C	Unaffected Unaffected 0 1 if a negative result is ge 1 if a zero result is genera 0 Unaffected	enerated, 0 otherwise ated, 0 otherwise						
Mode Bit	OVM	operation is not affected by	y OVM bit value.						
Cycles	1								
Example	LH0	R1, R2							
		Before Instruction	After Instruction						
	R1	ABCD EF12h	R1 ABCD EF12h						
	R2	1234 5678h	R2 FFFF EF12h						

Syntax	LHUw sr	c, dst						
Operands	<i>src</i> : register, direct, 16-bit immediate-, or indirect- addressing modes <i>dst</i> : register mode (any register in CPU primary-register file)							
Opcode								
-	31	2	24 23		1615		8 7	0
	1011	1 0 1 1	Н	G	dst		src	
Word Fields								
				G	<i>src</i> addressing	g modes	-	
				00	register (any rec CPU primary-re	gister in gister file)	_	
				01	direct			
				10	indirect			
				11	immediate			
							_	
				Н	<i>src</i> half-word		-	
				0	half-word 0 (LS	half-word)	_	
				1	half-word 1 (MS	S half-word)		
							_	
Operation	Unsigned	d half-woi	rd (0,	1) of	src ightarrow dst			
	w = half-	word to I	oad (0, 1)				
	1	0		= W C	lesignator			
Description	The spec the 16 LS mode is a performs the correa of the <i>ds</i>	tified half Bs of the specified sign exte sponding t register.	-word dstre and ension half-v	d of th egister a half n of th word	te <i>src</i> operand is r. The <i>src</i> half-wor -word 1 ($H = 1$) i te 16-bit value int value (0000h or F	unsigned a rd is unsign is selected, to a 32-bit v FFFh) is sto	and right-shifted i ed. When immedi , the LHw instruct /alue. Consequer pred into the 16 LS	nto ate tion tly, SBs

Status Bits	lf ST flags regis	If ST (SET COND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.							
	LUF LV UF N Z V C	Unaffected Unaffected 0 0 1 if a zero result is generated, 0 0 Unaffected	otherwise						
Cycles	1								
Mode Bit	OVM	operation is not affected by OVM	bit value.						
Example	LHUC R1 [R1, R2 Before Instruction ABCD EF12h	After Instruction						
	R2	1234 5678h]	R2 0000 EF12h						

Syntax	LSH src_count, dst						
Operands	<i>src_count:</i> general-addressing modes (G) <i>dst:</i> register (any register in CPU primary-register file)						
Opcode							
	31 24 2 0 0 0 1 0 1 1	3 G	1615 dst	8 7 src_count	0		
Word Fields							
	<u> </u>	G	<i>src</i> addressing	g modes			
	-	00	register (any reg CPU primary-re	gister in gister file)			
		01	direct				
		10	indirect				
		11	immediate				
Operation Description	$count = 7 \text{ LSBs of } sr$ If $count \ge 0$: $dst << count \rightarrow ds$ Else: $dst >> count \rightarrow$ The seven LSBs of the	c_coun st dst ne src_c	t count operand cor	nstitute the 2s-compler	ment-shift		
	<i>count.</i> If <i>count</i> is great <i>count.</i> Low-order bits out through the C (ca	ater thai s shifted arry) bit.	n 0, the <i>dst</i> opera I in are zero-filled	ind is left-shifted by the , and high-order bits a	e value of ire shifted		
	Logical left-shift:						
	$C \leftarrow dst \leftarrow 0$						
	If <i>count</i> is less than 0 operand. The high-o shifted to the right. L	, the <i>dst</i> rder bit ow-orde	tis right-shifted by s of the <i>dst</i> oper er bits are shifted	y the absolute value of rand are zero-filled as out through the C (ca	the <i>count</i> they are urry) bit.		
	Logical right-shift:						
	$0 \rightarrow dst \rightarrow C$						
	If <i>count</i> is 0, no shift	is perfo	rmed, and the C	(carry) bit is cleared to	o 0.		
	If <i>count</i> is greater tha –32, the C bit is clea	an 32, th red to 0	ne C (carry) bit ge	ets the LSB. If <i>count</i> is	less than		
			Assembly I	anguage Instructions	14-145		

	The <i>src_count</i> operand is is assumed to be an unsig	assumed to t gned integer	be a signed integer, and the <i>dst</i>	operand
Status Bits	If ST (SET COND) = 0 ar flags are modified. If ST (\$ registers.	nd the destin SET COND) :	ation register is R0–R11, the c = 1, they are modified for all de	condition stination
	LUFUnaffectedLVUnaffectedUF0NMSB of the outputZ1 if a zero output isV0CSet to the value of	s generated, the last bit s	0 otherwise hifted out. 0 for a shift <i>count</i> c	of O
Mode Bit	OVM operation is not affe	ected by OVN	<i>I</i> bit value.	
Cycles	1			
Example 1	LSH R4,R7			
	Before Instruct R4 018h R7 02ACh LUF 0 LV 0 UF 0 V 0 Z 0 V 0 C 0	ion 24	After Instruction R4 018h R7 0AC00 0000h LUF 0 LV 0 UF 0 UF 0 V 0 V 0 V 1 Z 0 V 1 C 0	24
Example 2	LSH *-AR5(IR0),R5	ion	After Instruction	
	AR5 80 9908h IR0 4h R5 00 12C0 0000h Data at 80 9904h 0FFF FFFF4h LUF 0 LV 0 UF 0 V 0 Z 0 V 0 C 0	-12	AR5 80 9908h IR0 4h R5 00 0001 2C00h Data at 80 9904h 0FFF FFFF4h LUF 0 LV 0 UF 0 V 0 Z 0 V 0 C 0	-12

Syntax	LSH3 src_count, src, dst	
Operands	src, src_count: both type 1 or type 2 three-operand addressing modes	
	dst: register mode (any register in CPU primary-register file)	
Opcode		
	Гуре 1	
	31 24 23 1615 8 7	0
	0 0 1 0 0 1 0 0 0 T dst src src_count	
	Гуре 2	
	31 24 23 1615 8 7	0
	0 0 1 1 0 1 0 0 0 T dst src src_count	Ť
Word Fields		
Word Ficius		
Туре 1		
	T src addressing modes src_count addressing modes	
	00 register mode (any CPU register) register mode (any CPU register)	
	01 indirect mode (disp = 0, 1, IR0, IR1) register mode (any CPU register)	
	10 register mode (any CPU register) indirect mode (disp = 0, 1, IR0, IR1)	
	11 indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1)	
Type 2		
<i>,</i> ,	T src addressing modes src count addressing modes	—
	00 register mode (any CPU register) 8-bit signed immediate	
	01 register mode (any CPU register) indirect mode *+ARn(5-bit unsigned displacement)	
	 indirect mode *+ARn(5-bit unsigned displacement) 8-bit signed immediate 	
	11 indirect mode *+ARn1(5-bit unsigned displacement) indirect mode *+ARn2(5-bit unsigned displacement)	d
Operation	$count = 7 \text{ LSBs of } src_count$ f count ≥ 0 : src << count \rightarrow dst Else: src >> count \rightarrow dst	

Description	The seven LSBs of the <i>src_count</i> operand constitute the 2s-complement shift <i>count</i> .
	If <i>count</i> is greater than 0, a copy of the <i>src</i> operand is left-shifted by the value of <i>count</i> , and the result is written to the <i>dst</i> (the <i>src</i> is not changed). Low-order bits shifted in are zero-filled, and high-order bits are shifted out through the C (carry) bit.
	Logical left-shift:
	$C \leftarrow src \leftarrow 0$
	If <i>count</i> is less than 0, the <i>src</i> operand is right-shifted by the absolute value of <i>count</i> . The high-order bits of the <i>dst</i> operand are zero-filled as shifted to the right. Low-order bits are shifted out through the C (carry) bit.
	Logical right-shift:
	$0 \rightarrow src \rightarrow C$
	If <i>count</i> is 0, no shift is performed and the C (carry) bit is set to 0.
	If <i>count</i> is greater than 32, the carry (C) bit is set to the LSB. If <i>count</i> is less than 32, the carry bit is cleared to 0. This also applies to LSH.
	The <i>src_count</i> operand is assumed to be a signed integer. The <i>src</i> and <i>dst</i> operands are assumed to be unsigned integers.
Status Bits	If ST (SET COND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.
	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero output is generated, 0 otherwise V 0 C Set to the value of the last bit shifted out. 0 for a shift <i>count</i> of 0
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

Syntax	LSH3 src_count, src2, dst1 STI src3, dst2								
Operands	$src_count: register (R0 - R7)$ src2: indirect (disp = 0, 1, IR0, IR1) dst1: register (R0 - R7) src3: register (R0 - R7) dst2: indirect (disp = 0, 1, IR0, IR1)								
Opcode									
	31 24 23 1615 8 7 0 1 1 0 1 1 1 0 dst1 src_count src3 dst2 src2								
Word Fields	None								
Operation	$\begin{array}{l} count = 7 \text{ LSBs of } src_count \\ \text{If } count \geq 0: \\ src2 << count \rightarrow dst1 \\ \text{Else:} \\ src2 >> count \rightarrow dst1 \\ src3 \rightarrow dst2 \end{array}$								
Description	The seven LSBs of the <i>src_count</i> operand constitute the 2s-complement shift <i>count</i> .								
	If <i>count</i> is greater than 0, a copy of the <i>src2</i> operand is left-shifted by the value of <i>count</i> and the result is written to <i>dst1</i> (<i>src2</i> is not changed). Low-order bits shifted in are zero-filled, and high-order bits are shifted out through the C (carry) bit.								
	Logical left-shift:								
	$C \leftarrow src2 \leftarrow 0$								
	If <i>count</i> is less than 0, a copy of the <i>src2</i> operand is right-shifted by the absolute value of <i>count</i> . The high-order bits of the <i>dst</i> operand are zero-filled as shifted to the right. Low-order bits are shifted out through the C (carry bit).								
	Logical right-shift:								
	$0 \rightarrow \textit{src2} \rightarrow C$								
	If <i>count</i> is 0, no shift is performed and the carry bit is set to 0.								
	The <i>src_count</i> operand is assumed to be a signed integer, and the <i>src2</i> and <i>dst1</i> operands are assumed to be unsigned integers. All registers are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation be-								
	Assembly Language Instructions 14-149								

	ing pe as inp	erformed in out the cont	parallel (LSF ents of the re	l3) writes to egister befo	o the s ore it i	ame register, th s modified by th	ien STI accepts ie LSH3.
	If src2	and <i>dst2</i> p	pint to the sa	me locatio	n, <i>srcź</i>	is read before t	he write to <i>dst2.</i>
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of th 1 if a zero 0 Set to the	d e output output is ge value of the	nerated, 0 last bit shi	other fted o	wise ut. 0 for a shift d	count of 0
Mode Bit	OVM	operation is	s affected by	OVM bit v	alue.		
Cycles	1						
Example 1]	LSH3 I STI I	R2,*++AR3 R4,*-AR5	(1) , R0			
	БА Г	Befor	e Instruction		Do	After Instr	ruction
		20090	3n] 2b]	24	R2		24
			211 Dh		R0		
		0.00	Ch	220	R4		220
	AR5	80 98A	3h	220	AR5	80 98A3h	220
	Data a	<u>t 80 98C3h</u> 0A0	 Ch		Data	at 80 98C3h 0ACh	
	Data a	t 80 98A2h (Dh		Data	at 80 98A2h 0DCh	220
			0		LUF	0	
			0		LV	0	
			0				
					N 7		
	ν Γ				۲ ۷		
			0		C	0	

14-150

Example 2		LSH3 R7	,*AR2(1),R2		
		STI RO),*+ARO(1)		
		Before	Instruction	After Instruction	
	R7	0FFFFF FF4h] –12	R7 0FFFFFFF4h	-12
	AR2	80 9863h]	AR2 80 9862h	
	R2	0h]	R2 2C000h	
	R0	12Ch	300	R0 12Ch	300
	AR0	80 98B7h]	AR0 80 98B7h	
	Data	at 80 9863h 2C00 0000h]	Data <u>at 80 9863h</u> 2C00 0000h	
	Data	<u>at 80 98B9h</u> 0h]	Data <u>at 80 98B8h</u> 12Ch	300
	LUF	0]	LUF 0	
	LV	0]	LV 0	
	UF	0]	UF 0	
	Ν	0]	N 1	
	Z	0]	Ζ0	
	V	0]	V 0	
	С	0]	C 0	

LWLct Load Word Left-Shifted

Syntax	LWLct src, dst					
Operands	ct: src: dst:	the count of bytes {0, 1, 2, or 3} to shift left ($ct \times 8 = shift$ in bits) register, direct, 16-bit immediate-, or indirect- addressing modes register mode (any register in CPU primary-register file)				

Opcode

_3	31	24 23			24 23			3	16	15 8 7	0	
•	1	0	1	1	0	1	0	В	G	dst	src	

Word Fields

G	src addressing modes
00	register (any register in CPU primary register file)
01	direct
10	indirect
11	immediate

В	<i>src</i> byte
00	no shift
01	shift left 1 byte space
10	shift left 2-byte spaces
11	shift left 3-byte spaces

Operation $src \ll \{0, 1, 2, \text{ or } 3\}$ bytes and merged with $dst \rightarrow dst$

Description The *src* operand is left-shifted the specified number of bytes and merged with the bytes of the *dst* register that are below the left-shifted LSB of the *src* operand. When immediate mode is selected, this instruction performs a sign extension of the 16-bit immediate value into a 32-bit value; then, this 32-bit value is shifted and merged.

Status Bits	lf ST (flags a registe	SET COND) = 0 and the destinatio are modified. If ST (SET COND) = 1 ers.	n reg , they	ister is R0 – R11, the condition are modified for all destination
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the output 1 if a zero result is generated, 0 o 0 Unaffected	therw	rise
Mode Bit	OVM	operation is not affected by OVM b	oit valu	le.
Cycles	1			
Example	LWL2 R1 [R2 [R1, R2 Before Instruction ABCD EF12h 1234 5678h	R1 R2	After Instruction ABCD EF12h EF12 5678h

LWRct Load Word Right-Shifted

Syntax	LWRc	t src, dst		
Operands	ct : src: dst:	the count of bytes {0, 1, 2, or 3} to shift right (ct \times 8 = shift in bits) register, direct, 16-bit immediate-, or indirect-addressing modes register mode (any register in CPU primary-register file)		

Opcode

3	1		24 23			3	16	15 8 7	0		
1	0	1	1	0	1	1	В	G	dst	src	

Word Fields

G	src addressing modes
00	register (any register in CPU primary-register file)
01	direct
10	indirect
11	immediate

В	<i>src</i> byte
00	no shift
01	shift left 1 byte space
10	shift left 2-byte spaces
11	shift left 3-byte spaces

Operation $src \gg \{0, 1, 2, \text{ or } 3\}$ bytes and merged with $dst \rightarrow dst$

Description The *src* operand is right-shifted the specified number of bytes and merged with the bytes of the *dst* register that are above the right-shifted MSB of the *src* operand. Sign is not extended. When immediate mode is selected, this instruction performs a sign extension of the 16-bit immediate value into a 32-bit value; then, this 32-bit value is shifted and merged.

Status Bits	If ST (flags a registe	SET COND) = 0 and the destinatic are modified. If ST (SET COND) = 1 ers.	on register is R0 – R11, the condition , they are modified for all destination
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the output 1 if a zero result is generated, 0 o 0 Unaffected	otherwise
Mode Bit	OVM o	operation is not affected by OVM b	bit value.
Cycles	1		
Example	LWR1 AR1 A R2	AR1, R2 Before Instruction BCD EF12hEF 1234 5678h	After Instruction AR1 ABCD EF12h R2 12AB CDEFh

MBct Merge Byte, Left-Shifted

Syntax	MBct src, dst
Operands	ct:the count of bytes $\{0, 1, 2, 3\}$ to shift left (ct $\times 8$ = shift in bits)src:register-, direct-, or indirect-addressing modesdst:register mode (any register in CPU primary-register file)
A 1	

Opcode

31	24 23	3	16	15 8 7
1011	100 B	G	dst	src

Word Fields

G	src addressing modes
00	register (any register in CPU primary-register file)
01	direct
10	indirect
11	immediate

В	<i>src</i> byte
00	no shift
01	shift left 1 byte space
10	shift left 2-byte spaces
11	shift left 3-byte spaces

Operation 8 LSBs of *src* << $\{0, 1, 2, \text{ or } 3\}$ bytes and merged with $dst \rightarrow dst$

Description The eight LSBs of the *src* operand are left shifted 0, 1, 2, or 3 bytes and merged with the bits of the *dst* register that are below the left-shifted LSB of the *src* operand. When immediate mode is selected, this instruction performs a sign extension of the 16-bit immediate value into a 32-bit value; then, this 32-bit value is shifted and merged.

Status Bits	If ST (flags a registe	SET COND) = 0 ure modified. If S ers.	and the destination T (SET COND) = ⁻	on register is R0 – R11, the condition 1, they are modified for all destination
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the outp 1 if a zero resul 0 Unaffected	out t is generated, 0 d	otherwise
Mode Bit	OVM	operation is not a	affected by OVM	bit value.
Cycles	1			
Example	MB2	AR1, AR2 Before Instr ABCD EF12h 1234 5678h	ruction (0012 0000h)	After Instruction AR1 ABCD EF12h AR2 1212 5678h

MHct Merge Half-Word, Left-Shifted

Syntax	MHct	src, d	st						
Operands	ct: src: dst:	the co registe registe	unt of er-, di er mo	f hal rect- de (a	f-word -, 16-bi any reç	(16-bit) t immeo gister in	shifts diate-, or indirect-ac CPU primary-regis	ldressing modes ter file)	
Opcode									
	31		24	23		16	15	8 7	0
	101	111	00	Н	G	dst		src	
Word Fields									
				_	G	<i>src</i> ad	dressing modes	-	
					00	registe CPU p	er (any register in primary-register file)	_	
					01	direct			
					10	indired	t		
					11	immec	liate	_	
					н	<i>src</i> ha	lf-word	-	
					0	half-w	ord 0 (LS half-word)	—	
					1	half-w	ord 1 (MS half-word)		
Operation	16 LSI	Bs of s	rc <<	{0.	1} half-	words a	and merged with <i>ds</i>	$-$ t \rightarrow dst	

- **Description** The 16 LSBs of the *src* operand are left shifted 0 or 1 half-words and merged with the bits of the *dst* register that are below the left-shifted LSB of the *src* operand. When immediate mode is selected, this instruction performs a sign extension of the 16-bit immediate value into a 32-bit value; then, this 32-bit value is shifted and merged.
- **Status Bits** If ST (SET COND) = 0 and the destination register is R0–R11, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.
 - LUF Unaffected
 - LV Unaffected
 - **UF** 0
 - **N** MSB of the output
 - Z 1 if a zero result is generated, 0 otherwise
 - **V** 0
 - C Unaffected

Mode Bit	$\ensuremath{\text{OVM}}$ operation is not affected by $\ensuremath{\text{OVM}}$ bit v	alue.
Cycles	1	
Example	MH1 AR1, AR2 Before Instruction	After Instruction
	AR1 ABCD EF12h (EF12 0000h) AF AR2 1234 5678h AF	ABCD EF12h R1 ABCD EF12h R2 EF12 5678h

- ,	MPYF src, dst								
Operands	<i>src:</i> general-ad <i>dst</i> : register (Re	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (R0 – R11)							
Opcode									
	_312	24 23	1615	8 7 0					
	00001010	0 G	dst	src					
Word Fields									
		G	<i>src</i> addressing m	odes					
		00	register (R0–R11)						
		01	direct						
		10	indirect						
		11	immediate						
Operation	$\textit{dst} \times \textit{src} \rightarrow \textit{dst}$								
Description	The product of the	dstand sr	coperands is loaded $(B0-B11)$ and details	d into the <i>dst</i> register. The val-					
	ues at <i>src</i> (if in regi sion floating-point precision floating-	t numbers. point numl	For nonregister mo	ode, <i>src</i> is treated as single-					
Status Bits	ues at <i>src</i> (if in regision floating-point precision floating- LUF 1 if a floatin LV 1 if a floatin UF 1 if a floatin N 1 if a nega Z 1 if a zero V 1 if a floatin C Unaffected	t numbers. point numl ng-point ur ng-point ov ng-point ur tive result i result is ge ng-point is d.	For nonregister mo per. derflow occurs, und reflow occurs, unch derflow occurs, 0 o s generated, 0 otherwis overflow occurs, 0 o	changed otherwise nanged otherwise therwise erwise se otherwise					
Status Bits Mode Bit	ues at <i>src</i> (if in regision floating-point precision floating- LUF 1 if a floatin LV 1 if a floatin UF 1 if a floatin N 1 if a negar Z 1 if a zero V 1 if a floatin C Unaffected OVM operation is	numbers. point numl ng-point ur ng-point ov ng-point ur tive result i result is ge ng-point is d. not affecte	For nonregister mo per. derflow occurs, und erflow occurs, unch derflow occurs, 0 o s generated, 0 otherwis overflow occurs, 0 ed by OVM bit value	changed otherwise nanged otherwise therwise erwise se otherwise					

Multiply Floating-Point Value MPYF

Example

MPYF R0,R2



After Instruction

R0 07 0C80 0000h 1.4050e + 02 R2 0A 600F 2000h 1.79247266e + 03 LUF 0 0 0 0 0 0

0

LV

UF

Ν

Ζ

۷

С

Assembly Language Instructions 14-161

Syntax	MP	/F3 src2, si	rc1, ds	t					
Operands	src1 dst:	, <i>src2:</i> type regis	1 or ty ster mo	pe 2 th de (R	nree-opera) – R11)	nd addressi	ng modes		
Opcode									
	Тур	e 1							
	31		24 23		16	15	8 7	0	
	0 0	1 0 0 1 0	0 1	Т	dst	src1		src2	
	Тур	e 2							
	31		24 23		16	15	8 7	0	
	0 0	1 1 0 1 0	0 1	Т	dst	src1		src2	
Word Fields									
турет		oro1	oddrog	oing p	odoo	0*02	oddroooin	a modoo	
	- 00	register m	adde (BC) R1		register m	ode (B0 —	B11)	
	01	indirect m	nde (dis	n = 0	' <i>)</i> I IR0 IR1)	register m	ode (no —	PLI register)	
	10	register m	nde (RC	P = 0, $P = R1^{\circ}$	1)	indirect m	ode (disp =)	0 1 IB0 IB1)	
	11	indirect m	ode (dis	α = 0.	., I. IR0. IR1)	indirect m	indirect mode (disp = $0, 1, 100, 101$)		
Type 2				,	, -, ,		(-	-,,,,,	
1900 2	т	src1	addres	sina m	odes	src2	addressin	a modes	
	01	register m	ode (RC) — R1	1)	indirect m displacem	ode *+ARn((5-bit unsigned	
	11	indirect mo displacem	ode *+A ent)	Rn1(5-	bit unsignec	indirect mo displacem	ode *+ARn2 lent)	(5-bit unsigned	
Operation	src1	\vee src2 \rightarrow	det						
Description	Tho		asi ara1 an	d cro?	ia loodod iy		alatan Tha	values at are1	
Description	src2	product of a	5101 all	u 3162		nn nn actro	MINCHAR I FIA		
	exte mod	nded-precisie, they are	d <i>src2</i> a sion flo assum	are in i ating-p ned to	register mo point numb pe single-p	nto the <i>dst</i> re ode (R0–R11 ers. If <i>src1</i> a precision floa	I)), and <i>dsi</i> I)), and <i>dsi</i> Ind <i>src2</i> are ating-point	t are treated as in nonregister numbers.	
Status Bits	exte mod LUF LV UF N Z V C	nded-precis le, they are 1 if a floa 1 if a floa 1 if a floa 1 if a floa 1 if a zen 1 if a zen 1 if a floa Unaffec	d <i>src2</i> a sion flo assum ating-pe ating-pe ating-pe gative r o resu ating-pe ted	are in r ating-p ned to oint un oint ov oint un result i lt is ge oint is	register mo point numb pe single-p derflow occ erflow occ derflow occ s generate nerated, 0 overflow o	nto the <i>ast</i> re ode (R0–R11 ers. If <i>src1</i> a precision floa curs, unchan curs, unchan curs, 0 othe d, 0 otherwise ccurs, 0 othe	I), and <i>dsi</i> and <i>src2</i> are ating-point nged other ged otherw rwise se erwise	t are treated as in nonregister numbers. wise vise	
Status Bits Mode Bit	exte mod LUF UF N Z V C OVM	In Sici and inded-precis le, they are 1 if a floa 1 if a floa 1 if a floa 1 if a cer 1 if a zer 1 if a floa Unaffec	d <i>src2</i> a sion flo assum ating-pe ating-pe gative r o resu ating-pe ted is not	are in r ating-p ned to oint un oint ov oint un result i lt is ge oint is affecte	egister mo point numb be single-p derflow occ erflow occ derflow occ s generate nerated, 0 overflow o	bit value.	ignitier. The l)), and <i>dsi</i> and <i>src2</i> are ating-point nged other ged otherw rwise se erwise	t are treated as in nonregister numbers. wise vise	
Status Bits Mode Bit Cycles	exte mod LUF LV UF N Z V C OVN 1	in sicr and inded-precis le, they are 1 if a floa 1 if a floa 1 if a floa 1 if a zer 1 if a zer 1 if a floa Unaffec	d <i>src2</i> a sion flo assum ating-pr ating-pr ating-pr gative n o resu ating-pr ted is not	are in r ating-r ned to oint un oint ov oint un result i lt is ge oint is affecte	register mo point numb be single-p derflow occ derflow occ derflow occ s generate nerated, 0 overflow o	bit value.	I), and <i>dsi</i> and <i>src2</i> are ating-point nged other ged otherw rwise se erwise	<i>t</i> are treated as in nonregister numbers. wise <i>t</i> ise	

14-162

Syntax	MPYF3 srcA, srcB, dst1 ADDF3 srcC, srcD, dst2					
Operands	$\left. \begin{array}{c} srcA \\ srcB \\ srcC \\ srcD \end{array} \right\}$ Any two must be indirect (disp = 0, 1, IR0, IR1), and any two must be register (R0 – R7).					
	dst1	register (<i>c</i> 0 = R0 1 = R1	11):			
	dst2	register (<i>c</i> 0 = R2 1 = R3	<i>12</i>):			
	src1 src2 src3 src4	register register indirect indirect	(R0 - R7) (R0 - R7) (disp = 0, 1) (disp = 0, 1)	, IR0, IR1 , IR0, IR1))	
	Р	parallel-ad	ddressing mo	odes (0 \leq	P ≤ 3)	
	Operation	(P Field)				
	00 01 10 11	src3 × src src3 × src src1 × src src3 × src	4, src1 + src. 1, src4 + src. 2, src3 + src 1, src2 + src	2 2 4 4		
Opcode						
	31	24 23	3	1	615 8	3 7 0
	1 0 0 0	0 0 P d1	d2 src1	src2	src3	src4
Word Fields	None					
Operation	srcA× srcC+	srcB ightarrow dstarrow dst	2			
Description	A floating parallel. A execute cy from a reg to the sam before it is	-point multip II registers a ycle. This me gister and the register, th s modified b	blication and are read at th eans that if or e operation b nen MPYF3 a y the ADDF3	a floating ne beginn ne of the p being perf accepts as 3.	g-point additior ning and loade arallel operatic ormed in paral s input the cont	n are performed in d at the end of the ons (MPYF3) reads lel (ADDF3) writes tents of the register

	You c sourc assign ries, c encoc order	can code any c e operands as I nment of the sc depending on th ded accordingly of operands in 2 and <i>dst2</i> point	ombination of addi ong as you code tw ource operands s <i>rc</i> , ne combination of a r. The assembler m commutative opera to the same locatior	ressir o as i A – <i>si</i> ddres ay, w ations	ng modes for the ndirect and two a rcD to the src1 – ssing modes use hen not significa to simplify proce	e four possible as register. The <i>src4</i> fields va- d; the P field is ant, change the essing.
Status Bits	LUF LV UF N Z V C	1 if a floating- 1 if a floating- 1 if a floating- 0 0 1 if a floating- Unaffected	point underflow occ point overflow occu point underflow occ point overflow occu	surs, urs, urs, urs, or	unchanged other nchanged otherw) otherwise otherwise	wise vise
Mode Bit	ονμ	operation is no	t affected by OVM I	oit va	lue.	
Cycles	1					
Example	M. II di	PYF3 *AR5++	(1),*AR1(IR()),R	0	
	11 11		n.J			
	11 21	Before Ins	struction		After Instr	uction
	AR5	Before Ins 80 98C5h	struction	AR5	After Instr	uction
	AR5	Before Ins 80 98C5h 80 98A8h	struction	AR5 AR1	After Instr 80 98C6h 80 98A4h	uction
	AR5 [AR1 [IR0 [Before Ins 80 98C5h 80 98A8h 4h	struction	AR5 AR1 IR0	After Instr 80 98C6h 80 98A4h 4h	uction
	AR5 [AR1 [IR0 [R0 [Before Ins 80 98C5h 80 98A8h 4h 0h	struction	AR5 AR1 IR0 R0	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 2200 0000h	uction .88867188e + 01
	AR5 [AR1 [IR0 [R0 [R5 [B7 [Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h	1.79750e + 02	AR5 AR1 IR0 R0 R5 B7	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h	uction .88867188e + 01 1.79750e + 02 1.4050e + 02
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h	1.79750e + 02 1.4050e + 02	AR5 AR1 IR0 R0 R5 R7 R3	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a	Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h at 80 98C5h 34C 0000h	1.79750e + 02 1.4050e + 02 1.2750e + 01	AR5 AR1 IR0 R0 R5 R7 R3 Data	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h at 80 98C5h 34C 0000h	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a	Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h at 80 98C5h 34C 0000h at 80 98A4h 111 0000h	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h 4 at 80 98C5h 34C 0000h a t 80 98A4h 111 0000h	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a Data a LUF [Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h 34C 0000h 34C 0000h at 80 98A4h 111 0000h 0	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data Data	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h at 80 98C5h 34C 0000h at 80 98A4h 111 0000h 0	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a LUF [LUF [LV [Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h 34C 0000h at 80 98A4h 111 0000h 0 0	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data Data LUF LV	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h 08 2020 0000h 08 2020 0000h at 80 98C5h 34C 0000h at 80 98A4h 111 0000h 0 0	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a LUF [LV [UF [Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h 34C 0000h at 80 98C5h 34C 0000h 0h 01 00 0 0 0 0 0 0 0 0 0 0 0	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data Data LUF LV UF	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h 08 2020 0000h 4 80 98C5h 34C 0000h 111 0000h 0 0 0 0 0 0 0 0 0	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a LUF [LV [UF [N [N]	Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h 34C 0000h at 80 98A4h 111 0000h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data Data LUF LV UF N	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h 08 2020 0000h 08 2020 0000h 111 0000h 111 0000h 0 0 0 0 0 0	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a Data a LUF [LV [UF [N [Z [X]	Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h 34C 0000h at 80 98C5h 34C 0000h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data Data LUF LV UF N Z	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h 08 2020 0000h at 80 98C5h 34C 0000h at 80 98A4h 111 0000h 0 0 0 0 0 0 0	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0
	AR5 [AR1 [IR0 [R0 [R5 [R7 [R3 [Data a LUF [LV [UF [N [Z [V [2]	Before Ins 80 98C5h 80 98A8h 4h 0h 07 33C0 0000h 07 0C80 0000h 0h 34C 0000h 111 0000h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1.79750e + 02 1.4050e + 02 1.2750e + 01 2.265625e + 0	AR5 AR1 IR0 R5 R7 R3 Data Data LUF LV UF N Z V	After Instr 80 98C6h 80 98A4h 4h 04 6718 000h 2 07 33C0 0000h 07 0C80 0000h 08 2020 0000h 08 2020 0000h 08 2020 0000h 08 2020 0000h 08 2020 0000h 034C 0000h 111 0000h 0 0 0 0 0 0 0 0 0 0 0 0 0	uction .88867188e + 01 1.79750e + 02 1.4050e + 02 3.20250e + 02 1.2750e + 01 2.265625e + 0

Syntax	M S	PYF3 src2 TF src3, ds	, src1, dst st2	1			
Operands	src1:register (R0 - R7)src2:indirect (disp = 0, 1, IR0, IR1) $dst1:$ register (R0 - R7)src3:register (R0 - R7) $dst2:$ indirect (disp = 0, 1, IR0, IR1)						
Opcode							
	31	1111	24 23 dst1	src1	16 src3	15 8 det2	7 0
			usii	3101	3100	0312	5102
Word Fields	None						
Operation	src src	1 imes src2 ightarrow 3 ightarrow dst2	dst1				
Description	A float lel. All cycle. ister a registe modifi	ting-point n registers a This mean and the ope er, then the ied by the I	nultiplication are read at s that if on ration beir STF acce MPYF3.	on and a t the begir e of the p ng perforr epts as in	floating- nning an arallel op med in pa put the c	ooint store are p d loaded at the perations (MPY arallel (STF) re contents of the	performed in paral- end of the execute (F3) writes to a reg- ads from the same register before it is
	lf <i>src2</i> to <i>dst</i>	? and <i>dst2</i> 2.	point to the	e same lo	ocation,	then <i>src2</i> is rea	ad before the write
Status Bits	 LUF 1 if a floating-point underflow occurs, 0 unchanged otherwise LV 1 if a floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise C Unaffected 						
Mode Bit	ονμ	operation i	s not affeo	cted by C	VM bit v	value.	
Cycles	1						

MPYF3||STF Parallel MPYF3 and STF

Example

Ν	MPYF3 *-AR2	(1),R7,R0			
2	STE R3,*AF	(1R0)		After Inc	truction
	Belore In	Istruction		Alterins	truction
AR2	80 982Bh		AR2	80 982Bh	
R7	05 7B40 0000h	6.281250e + 01	R7	05 7B40 0000h	6.281250e + 01
R0	0h		R0	0D 09E4 A000h	8.82515625e + 03
R3	08 6B28 0000h	4.7031250e + 02	R3	08 6B28 0000h	4.7031250e + 02
AR0	80 9860h		AR0	80 9858h	
IR0	8h		IR0	8h	
Data	at 80 982Ah		Data	a at 80 982Ah	
	70C 8000h	1.4050e + 02		70C 8000h	1.4050e + 02
Data	at 80 9860h		Data	a at 80 9860h	
	0h			86B28 0000h	4.7031250e + 02
LUF	0		LUF	0	
LV	0		LV	0	
UF	0		UF	0	
Ν	0		Ν	0	
Z	0		Z	0	
V	0		V	0	
С	0		С	0	

Syntax		 srcA, srcB, dst1 srcC, srcD, dst2 			
	srcA srcB srcC srcD	Any two must be indirec any two must be regis R7).	et (disp = 0, 1 ster (R0 –	, IR0, IR1), a	nd
Operands	None				
Opcode					
	31	24 23	1615	8 7	0
	1 0 0 0 0	1 P d1 d2 <i>src1</i>	src2	src3	src4
Word Fields	None				
Operation	srcA × s srcD – s	rcB → dst1 srcC → dst2			
	dst1	register (<i>d1</i>): 0 = R0 1 = R1			
	dst2	register (<i>d2</i>): 0 = R2 1 = R3			
	src1	register (R0 – R7)			
	src2	register $(R0 - R7)$			
	src3 src4	indirect (disp = $0, 1$ indirect (disp = $0, 1$, IR0, IR1) , IR0, IR1)		
	Р	parallel-addressing mo	des (0 ≤ P ≤	3)	
	Operation	(P Field)			
	00 01 10	$src3 \times src4$, $src1 - src2$ $src3 \times src1$, $src4 - src2$ $src1 \times src2$, $src3 - src4$	2 2 4		
	11	src3 × src1, src2 – src4	4		
Description	A floating-p in parallel. execute cyc from a regis to the same before it is	point multiplication and a All registers are read at cle. This means that if on ster, and the operation b register, then MPYF3 a modified by the SUBF3	a floating-poi the beginning le of the parall being perform .ccepts as inp	nt subtractior and loaded a lel operations ed in parallel ut the conten	n are performed at the end of the (MPYF3) reads (SUBF3) writes ts of the register

Status Bits	You can code any con source operands as lon assignment of the sour ries, depending on the encoded accordingly. T order of operands in co	nbination of addre g as you code two ce operands <i>srcA</i> combination of add he assembler may mmutative operation	ssing modes for the four possible as indirect and two as register. The - srcD to the $src1 - src4$ fields va- dressing modes used; the P field is y, when not significant, change the ons to simplify processing. rs, unchanged otherwise
	LV 1 if a floating-poi UF 1 if a floating-poi N 0 Z 0 V 1 if a floating-poi V 1 if a floating-poi	int overflow occurs int underflow occu	s, unchanged otherwise rs, 0 otherwise s, 0 otherwise
	C Unanected		
Mode Bit	OVM operation is not a	ffected by OVM bi	t value.
Cycles	1		
Example	MPYF3 R5,*++AR SUBF3 R7,*AR3- or MPYF3 *++AR7(I SUBF3 R7,*AR3-	7(IR1),R0 -(1),R2 R1), R5,R0 -(1),R2	
	Before Instru	iction	After Instruction
	R5 03 4C00 0000h	1.2750e + 01	R5 03 4C00 0000h 1.2750e + 01
	AR7 80 9904h		AR7 80 990Ch
	IR1 8h		IR1 8h
	R0 0h		R0 04 6718 0000h 2.88867188e + 01
	R7 07 33C0 0000h 1	./9/50e + 02	R/ 0733C00000h 1.79750e + 02
	B2 0h		$\begin{array}{c} \text{B2} 0.5 \text{ E300 0000h} -3.9250\text{e} + 0.1 \end{array}$
	Data at 80 990Ch		Data at 80 990Ch
	111 0000h	2.250e + 00	111 0000h 2.250e + 00
	Data <u>at 80 98B2h</u> 70C 8000h	1.4050e + 02	Data at 80 98B2h 70C 8000h 1.4050e + 02
	LUF 0		
	LV 0		LV 0
	UF 0		
	N0		N 0
			~V

Syntax	MPYI src, dst						
Operands	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)						
Opcode							
•	31		24 23	1615		8 7	0
	0 0 0	0 1 0 1	0 1 G	dst		SIC	
Word Fields							
			G	<i>src</i> add	ressing modes		
			00	register CPU pri	(any register in mary-register file)	_	
			01	direct			
			10	indirect			
			11	immedia	ate	_	
Operation	dst×:	src ightarrow dst					
Description	The part of the pa	roduct of th <i>st</i> operand is assume 32 LSBs c	ne <i>dst</i> and s s, when rea d to be a 6 of the result	<i>rc</i> operand ad, are assi 4-bit signed	s is loaded into th umed to be 32-bit d integer. The out	e <i>dst</i> register. T signed integer put to the <i>dst</i> re	⁻ he <i>src</i> [.] s. The egister
	Intege from t	er overflow he MSB of	occurs wh the 32-bit	en any of t output valu	he 32 MSBs of the	ne 64-bit result	differs
Status Bits	If ST (flags a registe	SET CON are modifie ers.	D) = 0 and d. If ST (SE	the destina T COND) =	tion register is R(= 1, they are modi	0 – R11, the co ified for all desti	ndition ination
	LUF LV UF N Z V C	Unchange 1 if an inte 0 1 if a neg 1 if a zero 1 if an inte Unaffecte	ed eger overflo ative result o result is g eger overflo ed	ow occurs, is generate enerated, (ow occurs,	unchanged other ed, 0 otherwise) otherwise 0 otherwise	rwise	
Mode Bit	OVM	operation i	is affected I	oy OVM bit	value.		
Cycles	1						

Example



The result overflows and R5 contains the 32 LSBs of the result. To obtain the 32 MSBs, use the MPYSHI3 or the MPYUHI3 instructions.

Syntax	MPYI3 src2, src1, dst					
Operands	<i>src1, src2</i> :type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)					
Opcode						
	Туре 1					
	31 24.23 16	15 8.7 0				
	0 0 1 0 0 1 0 1 0 T dst	src1 src2				
	Туре 2					
	31 24 23 16	15 8 7 0				
	0 0 1 1 0 1 0 1 0 T dst	src1 src2				
Word Fields						
Туре 1						
	T src1 addressing modes	src2 addressing modes				
	00 register mode (any CPU register)	register mode (any CPU register)				
	01 indirect mode (disp = 0, 1, IR0, IR1)	register mode (any CPU register)				
	10 register mode (any CPU register)	indirect mode (disp = 0, 1, IR0, IR1)				
	11 indirect mode (disp = 0, 1, IR0, IR1)	indirect mode (disp = 0, 1, IR0, IR1)				
Туре 2						
	T src1 addressing modes	<i>src2</i> addressing modes				
	00 register mode (any CPU register)	8-bit signed immediate				
	01 register mode (any CPU register)	indirect mode *+ARn(5-bit unsigned displacement)				
	<pre>10 indirect mode *+ARn(5-bit unsigned displacement)</pre>	8-bit signed immediate				
	<pre>indirect mode *+ARn1(5-bit unsigned displacement)</pre>	indirect mode *+ARn2(5-bit unsigned displacement)				
Operation	src1 imes src2 ightarrow dst					
Description	The product of the numbers at <i>src1</i> and a multiplied numbers are assumed to be a signed 64-bit integer. The least-significant bits of the result.	<i>src2</i> is loaded into the <i>dst</i> register. The 82-bit signed integers. The result is ashe output to the <i>dst</i> register is the 32				
	Integer overflow occurs when any of th from the MSB of the 32-bit <i>dst</i> value.	e 32 MSBs of the 64-bit result differs				
Status Bits	If ST (SET COND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.					
-------------	--					
	LUF Unchanged.					
	LV 1 if an integer overflow occurs, unchanged otherwise					
	UF 0					
	N 1 if a negative result is generated, 0 otherwise					
	Z 1 if a zero result is generated, 0 otherwise					
	V 1 if an integer overflow occurs, 0 otherwise					
	C Unaffected					
Mode Bit	OVM operation is affected by OVM bit value.					
Cycles	1					
Example	None					

Syntax	MPYI ADDI	3 srcA, srcB, dst1 3 srcC, srcD, dst2
Operands		
	srcA srcB srcC srcD	Any two must be indirect (disp = 0, 1, IR0, IR1), and any two must be register (R0 $-$ R7).
	dst1	register (<i>d1</i>): 0 = R0 1 = R1
	dst2	register (<i>d2</i>): 0 = R2 1 = R3
	src1 src2 src3 src4	register $(R0 - R7)$ register $(R0 - R7)$ indirect $(disp = 0, 1, IR0, IR1)$ indirect $(disp = 0, 1, IR0, IR1)$
	Р	parallel-addressing modes ($0 \le P \le 3$)
	Operation	(P Field)
	00 01 10 11	$src3 \times src4$, $src1 + src2$ $src3 \times src1$, $src4 + src2$ $src1 \times src2$, $src3 + src4$ $src3 \times src1$, $src2 + src4$
Opcode		
-	31	24 23 1615 8 7 <u>0</u>
	1 0 0 0	1 0 P d1 d2 src1 src2 src3 src4
Word Fields	None	
Operation	srcA× ∥ srcD+	$srcB \rightarrow dst1$ $srcC \rightarrow dst2$
Description	An integer registers a This mear and the op ister, then fied by the	r multiplication and an integer addition are performed in parallel. All are read at the beginning and loaded at the end of the execute cycle. Ins that if one of the parallel operations (MPYI3) reads from a register peration being performed in parallel (ADDI3) writes to the same reg- MPYI3 accepts as input the contents of the register before it is modi- e ADDI3.

	You c source assign ries, c encoc order	an code any combin e operands as long as nment of the source of lepending on the com led accordingly. The a of operands in comm	ation of add you code tw perands <i>src</i> bination of a assembler m utative oper	Iressin vo as in x <i>A – sr</i> addres nay, wl ations	ig modes for the ndirect and two a <i>rcD</i> to the <i>src1</i> – sing modes used hen not significat to simplify proce	e four possible is register. The <i>src4</i> fields va- d; the P field is nt, change the essing.
Status Bits	LUF LV UF N Z V C	Unchanged 1 if an integer overfle 0 0 1 if an integer overfle Unaffected	ow occurs, u ow occurs, C	inchar) other	nged otherwise wise	
Mode Bit	OVM	operation is affected	by OVM bit v	value.		
Cycles	1					
Example	MI Al	PYI3 R7,R4,R0 DDI3 *-AR3,*AR5-	-(1),R3			
		Poforo Instruction	-		After Instru	uction
	_	Belore Instruction	1		Alter Instit	
	R7 [14h	20	R7	14h	20
	R7 [R4 [14h 64h	20 100	R7 R4	14h	20 100
	R7 [R4 [R0 [14h 64h 0h	20 100	R7 R4 R0	14h 64h 07D0h	20 100 2000
	R7 [R4 [R0 [AR3 [14h 64h 0h 80 981Fh	20 100	R7 R4 R0 AR3	14h 64h 07D0h 80 981Fh	20 100 2000
	R7 [R4 [R0 [AR3 [AR5 [14h 64h 0h 80 981Fh 80 996Eh	20 100	R7 R4 R0 AR3 AR5	14h 64h 07D0h 80 981Fh 80 996Dh	20 100 2000
	R7 [R4 [R0 [AR3 [R3 [14h 64h 0h 80 981Fh 80 996Eh 0h	20 100	R7 R4 R0 AR3 AR5 R3	14h 64h 07D0h 80 981Fh 80 996Dh 0h	20 100 2000
	R7 [R4 [AR3 [AR5 [R3 [Data a	14h 64h 0h 80 981Fh 0h 80 996Eh 0h t 80 981Eh 0FFFF FFCBh	20 100 -53	R7 R4 R0 AR3 AR5 R3 Data	14h 64h 07D0h 80 981Fh 80 996Dh 0h at 80 981Eh 0FFFF FFCBh	20 100 2000 -53
	R7 [R4 [R0 [AR3 [R3 [Data a Data a	14h 14h 64h 0h 80 981Fh 80 996Eh 0h t 80 981Eh 0FFFF FFCBh t 80 996Eh 35h	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data	14h 14h 64h 07D0h 80 981Fh 80 996Dh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h	20 100 2000 53 53
	R7 [R4 [R0 [AR3 [R3 [Data a Data a LUF [14h 14h 64h 0h 80 981Fh 0h 80 996Eh 0h 180 981Eh 0FFFF FFCBh 180 996Eh 35h 0	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data LUF	14h 14h 64h 07D0h 80 981Fh 80 996Dh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h 0	20 100 2000 -53 53
	R7 [R4 [R0 [AR3 [R3 [Data a LUF [LV [14h 64h 0h 80 981Fh 0h 80 996Eh 0h t 80 981Eh 0FFFF FFCBh t 80 996Eh 35h 0 0	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data LUF LV	14h 64h 07D0h 80 981Fh 80 996Dh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h 0 0	20 100 2000 -53 53
	R7 [R4 [R0 [AR3 [R3 [Data a Data a LUF [LV [UF [14h 14h 64h 0h 80 981Fh 80 996Eh 0h t 80 981Eh 0FFFF FFCBh tt 80 996Eh 35h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data LUF LV UF	14h 14h 64h 07D0h 80 981Fh 80 996Dh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h 0 0 0 0 0 0 0 0 0 0 0 0 0	20 100 2000 53 53
	R7 [R4 [R0 [AR3 [R3 [Data a Data a LUF [LV [UF [N [14h 14h 64h 0h 80 981Fh 0h 80 996Eh 0h t 80 981Eh 0FFFF FFCBh t 80 996Eh 35h 0 0 0 0 0 0 0 0 0 0 0 0 0 0	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data LUF LV UF N	14h 14h 64h 07D0h 80 981Fh 80 996Dh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h 0 0 0 0 0	20 100 2000 -53 53
	R7 [R4 [R0 [AR3 [R3 [Data a Data a LUF [LV [UF [N [Z [14h 14h 64h 0h 80 981Fh 80 996Eh 0h t 80 981Eh 0FFFF FFCBh tt 80 996Eh 35h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data LUF LV UF N Z	14h 14h 64h 07D0h 80 981Fh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h 0 0 0 0 0 0 0 0 0 0 0	20 100 2000 -53 53
	R7 [R4 [R0 [AR3 [R3 [Data a Data a LUF [LV [UF [N [Z [V]	14h 14h 64h 0h 80 981Fh 0h 80 996Eh 0h t80 981Eh 0FFFF FFCBh t80 996Eh 35h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	20 100 -53 53	R7 R4 R0 AR3 AR5 R3 Data Data LUF LV UF N Z V	14h 14h 64h 07D0h 80 981Fh 0h at 80 981Eh 0FFFF FFCBh at 80 996Eh 35h 0	20 100 2000 53 53

Syntax	MPYI3 STI	src2, src1, ds src3, dst2	st1			
Operands	<i>src1:</i> regist <i>src2:</i> indire <i>dst1:</i> regist <i>src3:</i> regist <i>dst2:</i> indire	er (R0 – R7) oct (disp = 0, 1, er (R0 – R7) er (R0 – R7) oct (disp = 0, 1,	IR0, IR1) IR0, IR1))		
Opcode						
	31 1 1 1 0 0	24 23 0 0 <i>dst1</i>	src1	161 src3	5 8 dst2	7 0 src2
Word Fields	None					
Operation	src1 imes src	$c2 \rightarrow dst1$ st2				
Description	An integer multiplication and an integer store are performed in parallel. All reg- isters are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (MPYI3) writes to the same register, then STI accepts as input the contents of the register before it is modified by the MPYI3.					
	If src2 and d	st2 point to the	same loca	ation, <i>src</i>	2 is read befor	e the write to <i>dst2.</i>
	Integer over from the mo	flow occurs wh st significant b	ien any o it of the 3	f the 32 2-bit <i>dst</i>	MSBs of the 6 1 value.	64-bit result differs
Status Bits	LUF Unch LV 1 if ar UF 0 N 1 if a Z 1 if a V 1 if ar C Unaff	anged n integer overfl negative resul zero result is g n integer overfl ected	ow occur t is gener jenerated ow occur	s, uncha ated, 0 c I, 0 other s, 0 othe	nged otherwis therwise wise rwise	e
Mode Bit	OVM operat	ion is affected	by OVM	bit value.		
Cycles	1					

MPYI3||STI Parallel MPYI3 and STI3

Example

MPYI3 *++AR0(1), R5, R7 R2,*-AR3(1) ∥ STI

Before	Instruction
--------	-------------

80 995Ah AR0

R5

R7

R2

٧

С

32h 0h 0DCh

0

0

AR3 80 982Fh Data at 80 995Bh

0C8h

Data	at	80	982Eh

	at 00 00111	
		0h
LUF		0
LV		0
UF		0
Ν		0
Z		0

50

220

200

Data at 80 982Eh

Data at 80 995Bh

AR0

R5

R7

R2

AR3

LUF LV

Ν

200

220

50

10000

220

0
0

After Instruction

80 995Bh

32h

2710h

0DCh

0C8h

0DCh

80 982Fh

UF	0

0

- Ζ 0
- 0 ٧ 0
- С

Syntax	MPYI3 SUBI3	srcA, srcB, srcC, srcD,	dst1 dst2					
Operands	srcA srcB srcC srcD	Any two must b any two must b	be indirect be register	(disp = ((R0 – R	0, 1, IR 7).	0, IR1),	and	
	dst1	register (<i>d1</i>): 0 = R0 1 = R1						
	dst2	register (<i>d2</i>): 0 = R2 1 = R3						
	src1 src2 src3 src4	register (R register (R indirect (di indirect (di	0 – R7) 0 – R7) isp = 0, 1, isp = 0, 1,	IR0, IR1 IR0, IR1))			
	Р	parallel-addre	ssing mo	des (0 \leq	P ≤ 3)			
	Operation	(P Field)						
	00 01 10 11	$src3 \times src4$, si $src3 \times src1$, si $src1 \times src2$, si $src3 \times src1$, si	rc1 – src2 rc4 – src2 rc3 – src4 rc2 – src4)) ! !				
Opcode								
	31 1 0 0 0 1	24 23 1 P d1 d2	src1	16 src2	615 s	8 rc3	7 s	0 Src4
Word Fields	None							
Operation	srcA × s srcD – s	rcB ightarrow dst1 rcC ightarrow dst2						
Description	An integer All register cycle. This register an same regis it is modifie	multiplication a s are read at th means that if o d the operation ter, then MPYIC ed by the SUBI	and an intended beginn one of the n being person 3 accepts 3.	eger subt ing and parallel erformed as input t	traction loaded operati in para the con	are per at the e ons (MF allel (SL tents of	formed end of th YI3) re JBI3) wi the regi	in parallel. ne execute ads from a rites to the ster before
	You can co source ope	ode any combi erands as long a	ination of as you coo	address de two as	sing mo s indired	odes for ct and tv	the fou vo as re	ır possible gister. The

	assignment of the source operands <i>srcA</i> – <i>srcD</i> to depending on the combination of addressing mo coded accordingly. The assembler may, when not of operands in commutative operations in order t	the <i>src1– src4</i> fields varies, des used; the P field is en- significant, change the order o simplify processing.
	Integer overflow occurs when any of the 32 MSE from the MSB of the 32-bit output value.	s of the 64-bit result differs
Status Bits	 LUF Unchanged LV 1 if an integer overflow occurs, unchanged UF 1 if an integer underflow occurs, 0 otherwith N 0 Z 0 V 1 if an integer overflow occurs, 0 otherwish C Unaffected 	d otherwise se e
Mode Bit	OVM operation is affected by OVM bit value.	
Cycles	1	
Example	MPYI3 R2,*++AR0(1),R0 SUBI3 *AR5(IR1),R4,R2 or MPYI3 *++AR0(1),R2,R0 SUBI3 *AR5(IR1),R4,R2	
	Before Instruction	After Instruction
	R2 <u>32h</u> 50 R2 AB0 80 98E3h AB0	<u>320h</u> 800
	R0 0h R0 0	<u>01324h</u> 4900
	AR5 80 99FCh AR5	80 99F0h
	IR1 0Ch IR1	0Ch

07D0h

62h

4B0h

0

0

0

0

0

0

R4

LUF

LV

UF

N Z

V

С

Data at 80 98E4h

Data at 80 99FCh

 IR1
 OCh

 2000
 R4
 07D0h

 Data at 80 98E4h
 98
 62h

 1200
 Data at 80 99FCh
 1200

 LUF
 0

 LV
 0

 UF
 0

Ν	0
Ζ	0
V	0
С	0

2000

98

1200

Syntax	MPYS	SHI src, d	st				
Operands	src: (dst: 1	general-addr egister mod	ressing m e (any reę	odes gister in CP	U primary-registe	r file)	
Opcode							
	31	2	4 23	1615	8	8 7 C)
	0 0 0	1 1 1 0 1	1 G	dst	:	src	
Word Fields							
			G	<i>src</i> add	ressing modes	-	
			00	register CPU prir	(any register in mary-register file)	-	
			01	direct			
			10	indirect			
			11	immedia	te		
						_	
Operation	dst × .	src ightarrow dst					
Description	The 3 <i>dst</i> reg gers. registe LSBs	2 MSBs of th gister. These The result is er is the 32 of the result	ne product numbers assumed MSBs of	of the num , when reac to be a sigr the result.	bers at <i>dst</i> and <i>sr</i> d, are assumed to b ned 64-bit integer. ⁻ The MPYI instruct	c are loaded into the be signed 32-bit inte The output to the <i>ds</i> tion provides the 33	e :- st 2
Status Bits	If ST flags a registe	SET COND are modified. ers.) = 0 and If ST (SE	the destina T COND) =	ation register is R0 1, they are modifi)–R11, the condition ed for all destination	n n
	LUF LV UF N Z V	Unchanged Unchanged 0 1 if a negat 1 if all 64 bi 0	l ive result its of the p	is generate product are	ed, 0 otherwise 0, 0 otherwise		
	C	Unanected					
Mode Bit	ΟνΜ	operation is	not affect	ed by OVN	l bit value.		
Cycles	1						
Example	None						

Syntax	MPY	SHI3	src2,	src1	, dst					
Operands	 src1: type 1 or type 2 three-operand addressing modes src2: type 1 or type 2 three-operand addressing modes dst: register mode (any register in CPU primary-register file) 									
Opcode										
	Туре	e 1								
	31			24 23		1	615	8	3 7 0	
	0 0	1 0 1	0 0	0 1	Т	dst		src1	src2	
	Туре	2								
	31			24 23		1	615	8	3 7 0	
	0 0	1 1 1	0 0	01	Т	dst		src1	src2	
Word Fields										
Туре 1										
	Т	src1 a	ddress	sing I	mode	S		src2 addressing modes		
	00	registe	er mode	e (any	/ CPU	register)	1	register mode (any CPU register)		
	01 indirect mode (disp = 0, 1, IR0, IR1))	register mode (an	y CPU register)				
	10	registe	er mode	e (any	/ CPU	register)	i	indirect mode (dis	sp = 0, 1, IR0, IR1)	
	11	indired	ct mode	e (disp	0 = 0,	1, IR0, IR1)) i	indirect mode (dis	sp = 0, 1, IR0, IR1)	
Туре 2										
	Т	src1 a	ddress	sing I	mode	S	:	<i>src2</i> addressing	modes	
	00	registe	er mode	e (any	/ CPU	register)	8	8-bit signed imme	ediate	
	01	registe	er mode	e (any	/ CPU	register)	i	indirect mode *+A displacement)	Rn(5-bit unsigned	
	10 indirect mode *+ARn(5-bit unsigned displacement) 8-bit signed immediate					ediate				
	11	indireo displa	ct mode cement	e *+AF :)	Rn1(5∙	bit unsigne	edi d	indirect mode *+A displacement)	Rn2(5-bit unsigned	
Operation	src1	× src2	$2 \rightarrow ds$	t						
Description	The <i>dst</i> r 32-b outp prov	produc egiste it signe ut to th ides th	ct of the r. The i red integ ne <i>dst</i> i ne 32 L	e nur numt gers. regist .SBs	mbers pers a The r ter is of the	at the <i>src1</i> to the <i>src1</i> esult is ass the 32 MSI e result.	and and sum Bs c	nd <i>src2</i> operand I <i>src2</i> operands ed to be a signed of the result. The	ds is loaded into the are assumed to be d 64-bit integer. The e MPYI3 instruction	

Status Bits	If ST (SET COND) = 0 and the destination register is R0–R11, the condition flags are modified. If ST (SET COND) = 1, they are modified for all destination registers.						
	LUF Unchanged						
	LV 1 if an integer overflow occurs, unchanged otherwise						
	UF 0						
	N 1 if a negative result is generated, 0 otherwise						
	Z 1 if a zero result is generated, 0 otherwise						
	V 1 if an integer overflow occurs, 0 otherwise						
	C Unaffected						
Mode Bit	OVM operation is not affected by OVM bit value.						
Cycles	1						
Example	None						

Syntax	MPYUHI src,	dst				
Operands	<i>src:</i> general-addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)					
Opcode						
	31	24 23	1615	8 7	0	
	000111	100 G	dst	SrC		
Word Fields						
		G	<i>src</i> addres	sing modes		
		00	register (an CPU primar	y register in y-register file)		
		01	direct			
		10	indirect			
		11	immediate			
Operation Description	$dst \times src \rightarrow ds$ The 32 MSBs of into the dst reg 32-bit integers output to the of provides the 3	et of the produc jister. These a. The result dst register is 2 LSBs of th	t of the numbers numbers, wher is assumed to s the 32 MSBs e result.	s at <i>dst</i> and <i>src</i> ope read, are assume be an unsigned 64 of the result. The	erands are loaded ed to be unsigned 4-bit integer. The MPYI instruction	
Status Bits	If ST (SET CO flags are modi registers.	ND) = 0 and fied. If ST (SI	the destination ET COND) = 1,	n register is R0 – R they are modified t	11, the condition for all destination	
	LUF Unchar LV Unchar UF 0 N 0 Z 1 if all 6 V 0 C Unaffed	nged nged 64 bits of the cted	product are 0,	0 otherwise		
Mode Bit	OVM operation	n is not affec	ted by OVM bi	t value.		
Cycles	1					
Example	None					
14-182						

Syntax	MPYUHI3 src2, src1, dst							
Operands	<i>src1, src2</i> : both type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)							
Opcode								
	Туре 1							
	31 24 23 161	5 87 0						
	0 0 1 0 1 0 0 1 0 T dst	src1 src2						
	Type 2	5 8 7 0						
	0 0 1 1 1 0 0 1 0 T dst	src1 src2						
Word Fields Type 1	T							
	I src1 addressing modes	src2 addressing modes						
	00 register mode (any CPU register)	register mode (any CPU register)						
	10 register mode (asp = 0, 1, IRO, IRT)	indirect mode (disp. 0.1. IP0. IP1)						
	11 indirect mode (disp = 0, 1, IB0, IB1)	indirect mode (disp = 0, 1, IR0, IR1) indirect mode (disp = 0, 1, IR0, IR1)						
Туре 2								
	T src1 addressing modes	src2 addressing modes						
	00 register mode (any CPU register)	8-bit signed immediate						
	01 register mode (any CPU register)	displacement)						
	<pre>10 indirect mode *+ARn(5-bit unsigned displacement)</pre>	8-bit signed immediate						
	11 indirect mode *+ARn1(5-bit unsigned displacement)	indirect mode *+ARn2(5-bit unsigned displacement)						
Operation	$src1 \times src2 \rightarrow dst$							
Description	The product of the numbers at the <i>src1</i> and <i>dst</i> register. The numbers at the <i>src1</i> and 32-bit signed integers. The result is assumed to the <i>dst</i> register is the 32 MS tion provides the 32 LSBs of the result.	and <i>src2</i> operands is loaded into the ad <i>src2</i> operands are assumed to be med to be an unsigned 64-bit integer. SBs of the result. The MPYI3 instruc-						

MPYUHI3	Multiply Unsigned Integer Producing 32 MSBs, 3	Operands
---------	--	----------

Status Bits	If ST (flags a registe	SET COND) = 0 and the destination register is $R0 - R11$, the condition remodified. If ST (SET COND) = 1, they are modified for all destination ers.
	LUF LV UF N Z V C	Unchanged Unchanged 0 0 1 if all 64 bits of the product are 0, 0 otherwise 0 Unaffected
Mode Bit	OVM o	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	NEGB src, dst						
Operands	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)						
Opcode							
	31 24 0 0 0 1 0 1 1	1 23 0 G	1615 dst	8 7 0 src			
Word Fields							
				_			
		G	src addressing modes	_			
		00	register (any register in CPU primary-register file)				
		01	direct				
		10	indirect				
		11	immediate				
Operation	$0 - src - C \rightarrow dst$						
Description	The difference of the into the <i>dst</i> register	ne 0 <i>, src</i> , a r. The <i>dst</i> :	and C operands, calculated and <i>src</i> are assumed to be	d as shown, is loaded e signed integers.			
Status Bits	If ST (SETCOND) flags are modified. registers.	= 0 and th If ST (SET	e destination register is R0 COND) = 1, they are modi) – R11, the condition fied for all destination			
	LUFUnaffectedLV1 if an integeUF0N1 if a negativeZ1 if a zero redV1 if an integeC1 if a borrow	er overflov ve result is esult is ger er overflov v occurs, 0	v occurs, unchanged other s generated, 0 otherwise nerated, 0 otherwise v occurs, 0 otherwise) otherwise	wise			
Mode Bit	OVM operation is a	affected by	OVM bit value.				
Cycles	1						

NEGB Negate Integer With Borrow

Example

NEGB	R5,R7

	Before Instruction
R5	0FFFF FFCBh
R7	0h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	1

-53

After Instruction

-53

52

R5	0FFFF FFCBh	
R7	34h	
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	1	

1

Syntax	NEGI	= src, dst						
Operands	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (R0 – R11)							
Opcode								
	31		24 23	3	1615		8 7	0
	000	0 1 0 1	1 1	G	dst		src	
Word Fields								
word rields								
			_	G	<i>src</i> addressir	ng modes	_	
				00	register (any r CPU primary–	egister in register file)	_	
				01	direct			
				10	indirect			
				11	immediate			
							_	
Operation	0 – si	rc ightarrow dst						
Description	The c <i>dst</i> ar	lifference nd <i>src</i> ope	of the rands	0 and s are ass	erc operands is sumed to be flo	loaded into ating-point r	the <i>dst</i> register. numbers.	The
Status Bits	LUF LV UF N Z V C	1 if a floa 1 if a floa 1 if a floa 1 if a neg 1 if a zer 1 if a floa Unaffect	ating-p ating-p ating-p gative o resu ating-p ed	point une point ove point une result is ult is ger point ove	derflow occurs, erflow occurs, t derflow occurs, s generated, 0 nerated, 0 othe erflow occurs, 0	, unchanged unchanged o , 0 otherwise otherwise rwise 0 otherwise	otherwise otherwise	
Mode Bit	OVM operation is not affected by OVM bit value.							

OVM operation is not affected by OVM bit value.

1

Cycles

NEGF Negate Floating-Point Value

Example

NEGF *++AR3(2), R1 Before Instruction

	Before Instruction
AR3	80 9800h
R1	05 7B40 0025h 6.28125006e + 01
Data	at 80 9802h
	70C 8000h 1.4050e + 02
LV	0
UF	0
Ν	0
Z	0
V	0
С	0



Syntax	NEGF ∥ STF	src2, dst1 src3, dst2				
Operands	<i>src2:</i> indir <i>dst1:</i> regi <i>src3:</i> regi <i>dst2:</i> indir	rec <i>t</i> (disp = 0, 1, ster (R0 – R7) ster (R0 – R7) rect (disp = 0, 1,	IR0, IR1 IR0, IR1)		
Opcode						
	31 1 1 1 0 0	24 23 0 1 <i>dst1</i>	000	16 src3	15 8 dst2	7 0 src2
Word Fields	None					
Operation	$0 - src2$ $src3 \rightarrow$	$2 \rightarrow dst1$ dst2				
Description	A floating-p All register cycle. This ister and th register, the dified by th	point negation a rs are read at the means that if on ne operation beir en STF accepts ne NEGF.	nd a float e beginni e of the pa ng perforn as input tl	ing-poir ng and arallel o ned in pa he conte	nt store are per loaded at the e perations (STF arallel (NEGF) ents of the regis	formed in parallel. end of the execute) reads from a reg- writes to the same ster before it is mo-
	If <i>src2</i> and	<i>dst2</i> point to the	same loca	ation, <i>sr</i>	c2 is read befor	re the write to <i>dst2.</i>
Status Bits	LUF 1 if LV 1 if UF 1 if Z 1 if V 1 if C Unable	a floating-point u a floating-point u a floating-point u a negative resul a zero result is g a floating-point o uffected	underflow overflow c underflow t is gener generated overflow c	occurs, occurs, occurs ated, 0 l, 0 othe occurs, 0	, 0 unchanged unchanged oth , 0 otherwise otherwise erwise 0 otherwise	otherwise erwise
Mode Bit	OVM opera	ation is not affec	ted by O	VM bit v	alue.	
Cycles	1					

NEGF||**STF** *Parallel NEGF and STF*

Example	NEGF *AR4(1),R7	
	STF R2,*++AR5(1)	
	Before Instruction	After Instruction
	AR4 80 98E1h	AR4 80 98E0h
	R7 0h	R7 05 84C0 0000h -6.281250e + 01
	R2 07 33C0 0000h 1.79750e + 02	R2 07 33C0 0000h 1.79750e + 02
	AR5 80 9803h	AR5 80 9804h
	Data <u>at 80 98E1h</u> 57 B40 0000h 6.281250e + 01	Data at 80 98E1h 57 B40 0000h 6.281250e + 01
	Data <u>at 80 9804h</u> 0h	Data at 80 9804h 733 C000h 1.79750e + 02
	LUF 0	LUF 0
	LV 0	LV 0
	UF 0	UF 0
	N 0	N 0
	Ζ 0	Z 0
	V 0	V 0
	C 0	C 0

Syntax	NEGI src, dst				
Operands	<i>src:</i> general-addressing modes (G) <i>dst:</i> register (any register in CPU primary-register file)				
Opcode					
	_3124	23	1615	8 7	0
	0 0 0 0 1 1 0 0	0 G	dst	STC	
Word Fields					
	•	G	<i>src</i> addressing mod	les	
		00	register (any register CPU primary-register	in r file)	
		01	direct		
		10	indirect		
		11	immediate		
Operation	$0 - src \rightarrow dst$				
Description	The difference of th dst and src operand	ne 0 and a ds are as	<i>src</i> operands is loade sumed to be signed i	d into the <i>dst</i> register. T ntegers.	⁻ he
Status Bits	If ST (SETCOND) = flags are modified. I registers.	= 0 and th If ST (SE ⁻	ne destination register TCOND) = 1, they are	is R0 – R11, the condit modified for all destinat	ion ion
	LUFUnaffectedLV1 if an integerUF0N1 if a negativeZ1 if a zero reeV1 if an integerC1 if a borrow	er overflov ve result i sult is ge er overflov occurs, (w occurs, unchanged s generated, 0 otherw nerated, 0 otherwise w occurs, 0 otherwise 0 otherwise	otherwise vise	
Mode Bit	OVM operation is a	iffected by	y OVM bit value.		

Cycles

1

NEGI Negate Integer

Example

NEGI	174 , R5	(174	=	0AEh)
	Befo	re Instr	ucti	on
R5	0D	Ch		220
LUF		0		
LV		0		
UF		0		
Ν		0		
z [0		
V		0		
С		0		

After Instruction



Syntax	NEGI src2, STI src3, d	dst1 st2				
Operands	<i>src2:</i> indirect <i>dst1:</i> register <i>src3:</i> register <i>dst2:</i> indirect	(disp = 0, 1 (R0 - R7) (R0 - R7) (disp = 0, 1	, IR0, IR1 , IR0, IR1)		
Opcode						
	31 1 1 1 0 0 1	24 23 0 dst1	001	16 src3	15 8 dst2	3 7 0 src2
Word Fields	None					
Operation	$0 - src2 \rightarrow dst$ src3 $\rightarrow dst2$	1				
Description	An integer negation and an integer store are performed in parallel. All registers are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (NEGI) writes to the same register, then STI accepts as input the contents of the register before it is modified by the NEGI.					
	If src2 and dst2	point to the	same loc	ation, <i>si</i>	rc2 is read befo	re the write to <i>dst2.</i>
Status Bits	LUF Unaffec LV 1 if an in UF 0 N 1 if a ne Z 1 if a ze V 1 if an in C 1 if a bo	ted nteger overf gative resul ro result is g nteger overf rrow occurs	low occur It is gener generated low occur s, 0 otherv	rs, unch rated, 0 d, 0 othe rs, 0 oth wise	anged otherwis otherwise erwise erwise	se
Mode Bit	OVM operation	n is affected	by OVM	bit valu	е.	
Cycles	1					

NEGI||**STI** Parallel NEGI and STI

Example	NEGI *-AR3,R2			
	Before Instruction		After Instruction	
	AR3 80 982Fh		AR3 80 982Fh	
	R2 19h	25	R2 0FFFF FF24h	-220
	AR1 80 98A5h		AR1 80 98A6h	
	Data at 80 982Eh	220	Data at 80 982Eh	220
	Data at 80 98A5h		Data at 80 98A5h	25
	LUF 0		LUF 0	
	LV 0		LV 0	
	UF 0		UF 0	
	N 0		N 1	
	Z 0		Z 0	
	V 0		V 0	
	C 0		C 1	

Syntax	NOP src			
Operands	<i>src</i> : general-addr	essin	g modes (G)	
Opcode				
	31 24	23	1615	8 7 0
	0 0 0 0 1 1 0 0	1 G	dst	STC
Word Fields				
	•	G	src addressing mode	es
	-	00	register (no operation))
		10	indirect (modify ARn,	$0 \le n \le 7)$
	-			
Operation	No ALU or multiplie ARn is modified if <i>s</i>	r ope <i>rc</i> is s	rations. specified in indirect m	ode.
Description	If the <i>src</i> operand is operation is perform is omitted, no operation	s spea ied, a ation i	cified in the indirect m nd a dummy memory s performed.	node, the specified addressing read occurs. If the <i>src</i> operand
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected			
Mode Bit	OVM operation is n	ot affe	ected by OVM bit valu	Je.
Cycles	1			
Example 1	NOP			
	Before Ir	nstruct	ion	After Instruction
	PC 3Ah		PC [3Bh
Example 2	NOP *AR3(1)			
•	Before Ir	nstruct	ion	After Instruction
	PC 5h		PC [6h
	AR3 80 9900h		AR3	80 98FFh

Syntax	NORM src, dst			
Operands	<i>src:</i> general-addre <i>dst</i> : register (R0 –	essing mo R11)	odes (G)	
Opcode				
	31 24 2	3	1615	870
		G	asi	SIC
Word Fields				
	-	G	src addressing modes	-
	-	00	register (any register in	-
			CPU primary-register file)	
		01	direct	
		10	indirect	
	_	11	immediate	_
_				
Operation	norm (<i>src</i>) \rightarrow <i>dst</i>			
	example, the implied normalized <i>src</i> opera nent is set to the <i>src</i> sary to normalize the ting-point number.	bit is set and with t operand <i>src</i> . The	equal to the sign bit. The he implied bit removed. Th exponent minus the size o dst operand is assumed to	<i>dst</i> is set equal to the ne <i>dst</i> operand expo- of the left-shift neces- be a normalized floa-
	For values of <i>src</i> :			
	 If src (exp) = −12 If src (exp) = −12 For all other cas dst (man) is forc dst (man) = 0 an 	28 and <i>sr</i> 28 and <i>sr</i> ses of the ced to 0 d <i>dst (</i> ex	c (man) = 0, then dst = 0, c (man) ≠ 0, then dst = 0, e src, if a floating-point un and dst (exp) = -128. If p) = -128. Refer to Sectio	Z = 1, and $UF = 0$. Z = 0, and $UF = 1$. derflow occurs, then src (man) = 0, then n 5.7 on page 5-27.
Status Bits	LUF1 if a floating-LVUnaffectedUF1 if a floating-N1 if a negativeZ1 if a zero resV0	point und point und e result is ult is gen	erflow occurs, unchanged erflow occurs, 0 otherwise generated, 0 otherwise erated, 0 otherwise	l otherwise
	C Unaffected			
Mode Bit	OVM operation is no	t affected	I by OVM bit value.	
Cycles	1			

Normalize NORM

Example

NORM R1,R2

	Before Instruction	After Instruction
R1	04 0000 3AF5h	R1 04 0000 3AF5h
R2	07 0C80 0000h	R2 F2 6BD4 0000h 1.12451613e - 04
LUF	0	LUF 0
LV	0	LV 0
UF	0	UF 0
Ν	0	N 0
Z	0	Ζ 0
V	0	V 0
С	0	C 0

Syntax	NOT a	erc, dst					
Operands	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)						
Opcode							
	31	24	23	1615		8 7	0
	0 0 0	01101	1 G	dst		src	
Word Fields							
		-	G	<i>src</i> addr	essing modes	_	
		-	00	register (CPU prir	any register in nary-register file)	_	
			01	direct			
			10	indirect			
		_	11	immedia	te	_	
Operation	~src –	→ dst					
Description	The bi ter. Th The <i>d</i> :	twise-logical o e complemen st and <i>src</i> ope	compleme It is forme erands ar	ent of the s d by a logi e assumed	rc operand is loac cal NOT of each b d to be unsigned	ded into the <i>dst</i> reg bit of the <i>src</i> operar integers.	jis- nd.
Status Bits	If ST (flags a registe	SETCOND) = re modified. I ers.	= 0 and th f ST (SE⊺	e destinat ΓCOND) =	ion register is R0 1, they are modif	 R11, the conditi ied for all destinati 	ion ion
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the c 1 if a zero res 0 Unaffected	output sult is ge	nerated, 0	otherwise		
Mode Bit	OVM o	operation is a	ffected by	y OVM bit	value.		
Cycles	1						

Example

NOT @982Ch,R4

	Before Instruction
DP	80h
R4	Oh
Data	at 80 982Ch
	5E2Fh
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

	After Instruction				
DP	80h				
R4	0FFFF A1D0h				
Data at 80 982Ch					
	5E2Fh				
LUF	0				
LV	0				
UF	0				
Ν	1				
Z	0				
V	0				
С	0				

Syntax	NOT ∥ STI	src2, d src3, d	lst1 lst2					
Operands	<i>src2:</i> ind <i>dst1:</i> re <i>src3:</i> re <i>dst2:</i> ind	direct (dis gister (Ri gister (Ri direct (dis	sp = 0, 1, 0 - R7) 0 - R7) sp = 0, 1,	IR0, IR1 IR0, IR1)			
Opcode								
	31 1 1 1 0	0 1 1	24 23 dst1	000	16 <i>src3</i>	15 dst2	8 7	0 src2
Word Fields	None							
Operation	~src2 src3 –	→ dst1 → dst2						
Description	A bitwise-logical NOT and an integer store are performed in parallel. All regis- ters are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (NOT) writes to the same register, then STI accepts as input the contents of the register before it is modified by the NOT.							
	lf <i>src2</i> an	d <i>dst2</i> po	oint to the s	same loc	ation, <i>sr</i>	c2 is read be	efore the v	vrite to <i>dst2.</i>
Status Bits	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero result is generated, 0 otherwise V 0 C Unaffected 							
Mode Bit	OVM ope	eration is	not affec	ted by O	VM bit v	alue.		
Cycles	1							

	NOT	*+AR2,	R3				
	STI	R7,*	- AR4 (1	ER1)			
		Before In	struction			After Ins	truction
A	AR2	80 99CBh			AR2	80 99CBh	
F	73	0h			R3	0FFFF F3D0h	
F	77	0DCh		220	R7	0DCh	220
ŀ	AR4	80 9850h			AR4	80 9840h	
I	R1	10h			IR1	10h	
Γ	Data <u>at 80</u>) 99CCh 0C2Fh			Data	at 80 99CCh 0C2Fh	
[Data <u>at 80</u>) 9840h 0h			Data	at 80 9840h 0DCh	220
L	_UF	0			LUF	0	
L	V	0			LV	0	
ι	JF	0			UF	1	
1	N 🗌	0			Ν	0	
Z	<u>z</u>	0			Z	0	
١	/	0			V	0	
(0			С	0	

Example

Syntax	OR sro	c, dst						
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)							
Opcode								
	31	24	1 23	16	15	8 7	0	
	0 0 0	1 0 0 0 0	0 G	dst		SIC		
Word Fields								
			G	src a	ddressing modes			
			00	regist CPU	er (any register in primary-register file)			
			01	direct				
			10	indire	ct			
			11	imme	diate	_		
Operation	<i>dst</i> OF	$src \rightarrow dst$						
Description	The bit registe	wise-logical er. The <i>dst</i> ar	OR betwe nd <i>src</i> ope	en the s rands a	erc and <i>dst</i> operands are assumed to be u	s is loaded into the <i>d</i> insigned integers.	st	
Status Bits	If ST (flags a registe	SETCOND) re modified. ers.	= 0 and th If ST (SET	e destir FCOND	nation register is R0) = 1, they are modif	– R11, the condition fied for all destination	on on	
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the 1 if a zero re 0 Unaffected	output esult is ge	nerated	, 0 otherwise			
Mode Bit	OVM o	peration is r	not affecte	d by O	/M bit value.			
Cycles	1							

Example

OR *++AR1(IR1),R2

	Before Instruction
AR1	80 9800h
IR1	4h
R2	01256 0000h
Data	at 80 9804h 2BCDh
LUF	0
LV	0
UF	0
N	0
Z	0
V	0
С	0

After Instruction

_	
AR1	80 9804h
IR1 [4h
R2 [01256 2BCDh
Data a	at 80 9804h
	2BCDh
LUF [0
LV [0
UF [0
N [0
z [0
v [0

C 0

Syntax	OR3	src2, src1, dst					
Operands	<i>src1, src2</i> :type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)						
Opcode							
	Тур	e 1					
	31	24 23		161	5	8 7 0	
	0 0	1 0 0 1 0 1 1	Т	dst	src1	src2	
	Тур	e 2					
	31	24 23		161	5	8 7 0	
	0 0	1 1 0 1 0 1 1	Т	dst	src1	src2	
Word Fields							
Type 1							
	<u>т</u>	src1 addressing i	nodes	5	<i>src2</i> addressi	ng modes	
	00	register mode (any	/ CPU	register)	register mode ((any CPU register)	
	01	indirect mode (disp	0 = 0, -	I, IR0, IR1)	register mode ((any CPU register)	
	10	register mode (any	CPU	register)	indirect mode ((disp = 0, 1, IR0, IR1)	
	11	indirect mode (disp	0 = 0, 1	I, IR0, IR1)	indirect mode (disp = 0, 1, IR0, IR1)	
Туре 2							
	Т	src1 addressing i	nodes	5	<i>src2</i> addressi	ng modes	
	00	register mode (any	/ CPU	register)	8-bit signed im	mediate	
	01	register mode (any	CPU	register)	indirect mode * displacement)	+ARn(5-bit unsigned	
	10	indirect mode *+Al displacement)	Rn(5-b	it unsigned	8-bit signed im	mediate	
	11	indirect mode *+Al displacement)	Rn1(5-	bit unsigned	indirect mode * displacement)	+ARn2(5-bit unsigned	

Operation

src1 OR src2 \rightarrow dst

Description	The bi is load are as is sign	itwise-logical OR between the numbers at the <i>src1</i> and <i>src2</i> operands led into the <i>dst</i> register. The numbers at the <i>src1</i> , <i>src2</i> , and <i>dst</i> operands sourced to be unsigned integers. The <i>src2</i> immediate-addressing mode in extended.
Status Bits	If ST (ter is I registe	SETCOND) = 0, the condition flags are modified if the destination regis- R0 — R11. If ST (SETCOND) = 1, they are modified for all destination ers.
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the output 1 if a zero result is generated, 0 otherwise 0 Unaffected
Mode Bit	OVM	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	OR3 ∥ STI	src2, src3,	src1, dst dst2	1				
Operands	src1:register (R0 - R7)src2:indirect (disp = 0, 1, IR0, IR1) $dst1:$ register (R0 - R7)src3:register (R0 - R7) $dst2:$ indirect (disp = 0, 1, IR0, IR1)							
Opcode								
	31	1 0 0	24 23	cro1	16	15 8	7	0 1
		100	usti	SICT	SICO	0512	5102	
Word Fields	None							
Operation	src1 (src3 -)R <i>src2</i> – → <i>dst2</i>	→ dst1					
Description	A bitwise-logical OR and an integer store are performed in parallel. All regis- ters are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (OR3) writes to the same register, then STI accepts as input the contents of the register before it is modified by the OR3							
	lf <i>src2</i> ar	d <i>dst2</i> po	oint to the	same loc	ation, <i>sr</i>	c2 is read befor	e the write to <i>dsta</i>	2.
Status Bits	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero result is generated, 0 otherwise V 0 C Unaffected 							
Mode Bit		eration is	not affec	ted by O	VM bit v	alue.		
Cycles	1							

Exa	m	pl	e
			-

	OR3	*++AR2,R5,R2	2
	STI	R6,*AR1	
		Before Instruction	I
AR	2	80 9830h	
R5		80 0000h	
R2		0h	
R6		0DCh	220
AR	1	80 9833h	
Dat	ta <u>at 80 s</u>	9831h	
		9800h	
Dat	ta <u>at 80 s</u>	9883h	
		<u> </u>	
LUI	F	0	
LV		0	
UF		0	
Ν		0	
Ζ		0	
V		0	
С		0	

	After Instruction	
AR2	80 9831h	
R5	80 0000h	
R2	80 9800h	
R6	0DCh	
AR1	80 9882h	
Data at 80 9831h		
	9800h	
Data <u>at 80 9883h</u>		
	0DCh	
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

220

220
Syntax	POP dst													
Operands	dst: register (any register in CPU primary-register file)													
Opcode														
	31		24 23	0.1	16	15	0 0	0 0	0.0	8	7			0
	0000) 1 1 1	0 0	0 1	ast	0 0	0 0	0 0	0 0	0	0 0	000	0	0 0
Word Fields	None													
Operation	*SP	ightarrow dst												
Description	The top of the current system stack is popped and loaded into the 32 LSBs of the <i>dst</i> register. The top of the stack is assumed to be a signed integer. The POP is performed with a post decrement of the stack pointer. The eight MSBs (exponent) of an extended-precision <i>dst</i> register (R11–R0) are left unmodified. If required they can be recovered with a POPF instruction.													
Status Bits	If ST (SI flags are registers	ETCONI e modifie s.	D) = 0 ed. If S	and t T (SE	he desti TCONE	nation)) = 1,	regi they	ster are i	is R(nodi) — I ifiec	R11	, the all de	conc estin	dition ation
Mada Dit	LUF U LV U UF 0 N 1 Z 1 V 0 C U	Inaffecte Inaffecte if a neg if a zerc Inaffecte	ed ative r o resul ed	esult t is ge	is gener enerated	rated, d, 0 ot	0 oth herw	nerwi ise	se					
Mode Bit		eration	is not a	anect	ea by O		t vall	le.						
Cycles	1													
Example	POP R3	Befo	re Instr	uction	I				Aft	er Ir	nstru	ction		
	SP	80 985	56h				SP [80 9	855ł	ı			
	R3	012D	Ah		4 826		R3 [0FF	FF 0I	DA4ł	۱		-62	044
	Data at 8	0 9856h FFFF 0DA	\4h		-62 044		Data [at 80 0FF	98561 FF 01	า DA4l	۱		-62	044
	LUF LV UF Z V C						LUF [LV [UF [N [Z [V [C [

Syntax	POPF dst
Operands	<i>dst</i> : register (R0 – R11)
Opcode	
	31 24 23 1615 8 7 0 0 0 0 1 1 0 1 dst 0
Word Fields	None
Operation	*SP \rightarrow dst
Description	The top of the current system stack is popped and loaded into the <i>dst</i> register (32 MSBs). The eight LSBs of the <i>dst</i> register mantissa are set to 0. For this reason, POPF must be executed before the POP instruction when you are preserving the entire 40 register bits. The top of the stack is assumed to be a floating-point number. The POP is performed with a postdecrement of the stack pointer.
Status Bits	 LUF Unaffected UF 0 LV Unaffected N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 0 C Unaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	POPF R4
	$ \begin{array}{c c c c c c c c c c c c c c c c c c c $

Syntax	PUSH src						
Operands	src: register (ar	ny register	in CPU	primary-r	egister fil	e)	
Opcode	31 2	24 23	161	15		8 7	0
	0 0 0 0 1 1 1 1	0 0 1	src	0 0 0 0	0 0 0	00000	0000
Word Fields	None						
Operation	$\textit{src} \rightarrow {}^{*}\text{++SP}$						
Description	The contents of th stack. The intege (R0–R11) is saved with the PUSHF in PUSH is performe	e <i>src</i> regis er or man I with this in nstruction. ed with a p	ter (32 L tissa po nstructio The <i>sro</i> reincrer	SBs) are ortion of a on. The 8 N c is assun nent of th	pushed o an extend ISBs (exp ned to be e stack p	nto the cur ded-precisi conent) car a signed i ointer.	rent system on register t be pushed nteger. The
Status Bits	LUF Unaffected LV Unaffected UF Unaffected N Unaffected Z Unaffected V Unaffected C Unaffected						
Mode Bit	OVM operation is	not affect	ed by O'	VM bit va	ue.		
Cycles	1						
Example	PUSH R6						
	Before	Instruction			A1	ter Instructio	on
	SP 80 98AE	h] El	00 115	SP	80 9	98AFh	00.115
	R6 815B	n	33 115	Ro Data	at 80 984F	-h	33 115
		h		Data	[[[[[[[[[[[[[[[[[[[315Bh	33 115
	LUF	ס		LUF		0	
				LV		0	
		้อ					
	z	0		Z			
	V	 ס		V		0	
	C	ס		С		0	

Syntax	PUSHF src
Operands	<i>src:</i> register (R0 – R11)
Opcode	
	31 24 23 1615 8 7 0 0 0 0 0 1 1 1 1 1 0 1 src 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Word Fields	None
Operation	$src \rightarrow *++SP$
Description	The contents of the <i>src</i> register (32 MSBs) are pushed onto the current system stack. The <i>src</i> is assumed to be a floating-point number. The PUSH is performed with a preincrement of the stack pointer. The eight LSBs of the mantissa are not saved (notice the difference in R2 and the value on the stack in the example below), but they can be saved with the PUSH instruction. PUSHF should be executed after the PUSH instruction.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	PUSHF R2
	Before Instruction After Instruction SP 80 9801h SP 80 9802h R2 02 5C12 8081h 6.87725854e + 00 R2 02 5C12 8081h 6.87725854e + 00 Data at 80 9802h Data at 80 9802h LUF 0h UF 0 LV 0 LV 0 V 0 V 0 Z 0 V 0 C 0 V 0 C 0 V 0 C 0 V 0

Syntax	RCPF	src, dst					
Operands	src: dst:	extended-prec R0 – R11	ision re	gister-, dire	ct- and indirect-	addressing mod	les
Opcode							
	31	24 2	3	1615		8 7	0
	000		G	asi		SIC	
Word Fields							
			G	<i>src</i> addre	ssing modes	-	
		_	00	register (a CPU prima	ny register in ary-register file)	—	
			01	direct			
			10	indirect			
		_	11	immediate		_	
Operation	16-bit	reciprocal of <i>si</i>	$rc \rightarrow ds$	t			
Description	The 16 <i>dst</i> reç bers.	6-bit approxima gister. The <i>dst</i> a	tion of tl and <i>src</i> (ne reciproca operands ar	I of the <i>src</i> operative assumed to be	and is loaded int e floating-point	o the num-
Status Bits	 LUF 1 if a floating-point underflow occurs, unchanged otherwise LV 1 if a floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 is a zero result, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise C Unaffected 						
Mode Bit	OVM	operation is not	t affecte	d by OVM b	oit value.		
Cycles	1						
Example	None						

Syntax	RETIcond
Operands	None
Opcode	
	<u>31 24 23 1615 8 7 0</u>
	0 1 1 1 1 0 0 0 0 0 0 <i>cond</i> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Word Fields	None
Operation	If (<i>cond</i> is true) $(SP) \rightarrow PC$ $ST(PGIE) \rightarrow ST(GIE)$ $ST(PCF) \rightarrow ST(CF)$ Else, continue
Description	If the condition is true, then the top of the stack is popped to the PC, PGIE is copied to GIE, and PCF is copied to CF. If the condition is not true, then continue normal operation (see Section 14.2 on page 14-12 for a list of condition mnemonics, encoding, and flags).
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	4
Example	None

Syntax	RETIcondD
Operands	None
Opcode	
	<u>31 24 23 1615 8 7 0</u>
	0 1 1 1 1 0 0 0 0 1 <i>cond</i> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Word Fields	None
Operation	If (cond is true) $^{*}(SP) \rightarrow PC$ $ST(PGIE) \rightarrow ST(GIE)$ $ST(PCF) \rightarrow ST(CF)$ Else, continue
Description	Performs a delayed return from an interrupt or trap.
	Because this is a delayed return, the three instructions following the RETI <i>cond</i> D are fetched and executed. These three instructions should not modify the program flow, load the status register, or modify the stack pointer (SP) register. See Section 14.2 for a list of condition mnemonics, encoding, and flags.
	Interrupts are disabled for the duration of the RETI <i>cond</i> D.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

RETIcondD Return From Interrupt or Trap Conditionally Delayed

Syntax	RETScond	
Operands	None	
Opcode		
	31 24 23 1615	870
	0 1 1 1 1 0 0 0 1 0 0 <i>cond</i> 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0
Word Fields	None	
Operation	If <i>cond</i> is true: *SP \rightarrow PC. Else, continue.	
Description	A conditional return is performed. If the c is popped to the PC.	condition is true, the top of the stack
	The 'C4x provides 20 condition codes the (see Section 14.2 for a list of condition methods).	nat can be used with this instruction nemonics, encoding, and flags).
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected	
Mode Bit	OVM operation is not affected by OVM b	it value.
Cycles	4	
Example	RETSGE	
	Before Instruction	After Instruction
		PC 456h
	Data at 80 983Ch	Data at 80 983Ch
	456h	456h
	N 0	N 0
	z0	z 0
	V0	V 0
	C 0	C 0

Syntax	RND s	src, dst					
Operands	<i>src:</i> general-addressing modes (G) <i>dst:</i> register (R0 – R11)						
Opcode							
	31	24	23	1615	5	8 7	0
	0 0 0	100010	G	dst		src	
Word Fields							
		-	G	oro ada	traccing modes	_	
		-	<u>u</u>	Sic aut		_	
			00	CPU pr	imary-register file)		
			01	direct			
			10	indirect			
		_	11	immedi	ate		
Operation	rnd(<i>sr</i>	c) $ ightarrow$ dst					
Description	The re operar <i>src</i> op rounde not se	sult of roundin nd is rounded erand is exa ed to the most t the zero (z) s	ng the <i>si</i> to the ne ctly half positive status bi	rc operand earest sin way betw of those va t but the u	d is loaded into the gle-precision floati veen two single-pr alues. Notice that the underflow bit.	e <i>dst</i> register.The ing-point value. I recision values, he rounding of 0 c	e <i>src</i> f the it is loes
Status Bits	LUF LV UF	1 if a floating 1 if a floating 1 if a floating otherwise	-point ur -point ov -point ur	nderflow o verflow oc nderflow o	occurs, unchanged curs, unchanged o occurs or the <i>src</i> o	l otherwise otherwise perand is zero, 0	J
	N Z V C	1 if a negative Unaffected 1 if a floating	e result i -point ov	is generat verflow oc	ted, 0 otherwise curs, 0 otherwise		
Mode Bit		peration is af	ffactad h		t value		
MODE DI			nected b		t value.		
Cycles	1						

Example

RND R5,R2

	Before Instruction
R5	07 33C1 6EEFh 1.79755599e + 02
R2	0h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

After Instruction

R5	07 33C1 6EEFh 1.79755599e + 02
R2	07 33C1 6EEFh 1.79755600e + 02
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0

0 С

Ν Ζ V

Syntax	ROL dst					
Operands	dst: register (any register in CPU primary-register file)					
Opcode						
	31 24 23 1615 0 0 1 0 1 1 1 dst 0 0	8 7 0 0 0 0 0 0 0 0 0 1				
Word Fields	None					
Operation	dst left-rotated 1 bit \rightarrow dst					
Description	The contents of the <i>dst</i> operand are left-rotated one bit and loaded into the <i>dst</i> register. This is a circular rotate with the MSB transferred into the LSB.					
	Rotate left:					
		dst				
Status Bits	If ST (SETCOND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SETCOND) = 1, they are modified for all destination registers.					
	LUFUnaffectedLVUnaffectedUF0NMSB of the outputZ1 if a zero output is generated, 0 ofV0CSet to the value of the bit rotated of	otherwise out of the high-order bit				
Mode Bit	OVM operation is not affected by OVM b	it value.				
Cycles	1					
Example	Before Instruction R3 8002 5CD4h LUF 0 LV 0 UF 0 V 0 Z 0 V 0 Z 0 C 0	After Instruction R3 0004 B9A9h LUF 0 LV 0 UF 0 V 0 X 0 Z 0 V 0 C 1				

Syntax	ROLC dst									
Operands	dst: register (any register in CPU primary-register file)									
Opcode										
	<u>31</u> <u>24</u> <u>23</u> <u>1615</u> <u>8</u> <u>7</u> <u>0</u>									
Word Fields	None									
Operation	<i>dst</i> left-rotated 1 bit through carry bit \rightarrow <i>dst</i>									
Description	The contents of the <i>dst</i> operand are left-rotated one bit through the carry bit and loaded into the <i>dst</i> register. The MSB is rotated to the carry bit, at the same time the carry bit is transferred to the LSB.									
	Rotate left through carry bit:									
Status Bits	If ST (SETCOND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SETCOND) = 1, they are modified for all destination registers.									
	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero output is generated, 0 otherwise V 0 C Set to the value of the bit rotated out of the high-order bit 									
Mode Bit	OVM operation is not affected by OVM bit value.									
Cycles	1									
Example 1	ROLC R3									
	Before Instruction After Instruction R3 0000 0420h R3 00000 0841h LUF 0 LUF 0 LV 0 LV 0 V 0 UF 0 V 0 N 0 Z 0 Z 0 V 0 V 0 C 1 C 0									

ROLC Rotate Left Through Carry

Example 2

1.0 1.	1.0
	Before Instruction
R3	8000 4281h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

After Instruction

R3 [0000 8502h
LUF [0
LV [0
UF [0
Νſ	0

Syntax	ROR dst									
Operands	dst: register (any register in CPU primary-register file)									
Opcode										
	<u>31 24 23 1615 8 7 0</u>									
	$\begin{bmatrix} 0 & 0 & 0 & & 1 & 0 & 1 & 0 & 1 & & 1 & 1 & & dst \\ \end{bmatrix}$									
Word Fields	None									
Operation	dst right-rotated 1 bit $\rightarrow dst$									
Description	The contents of the <i>dst</i> operand are right-rotated one bit and loaded into the <i>dst</i> register. The LSB is rotated into the carry bit and also transferred into the MSB.									
	Rotate right:									
Status Bits	If ST (SETCOND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SETCOND) =1, they are modified for all destination registers.									
	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero output is generated, 0 otherwise V 0 C Set to the value of the bit rotated out of the high-order bit 									
Mode Bit	OVM operation is not affected by OVM bit value.									
Cycles	1									
Example	ROR R7									
	Before Instruction After Instruction R7 00000421h R7 80000210h LUF 0 LUF 0 LV 0 LV 0 UF 0 UF 0 V 0 VF 0 V 0 Z 0 V 0 V 0 C 0 C 1									

Syntax Operands Opcode	RORC <i>dst dst:</i> register (any register in CPU primary-register file)									
	31 24 23 1615 0 0 1 0 1 1 dst 1 1	8 7 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1								
Word Fields	None									
Operation	dst right-rotated 1 bit through carry bit \rightarrow	dst right-rotated 1 bit through carry bit $\rightarrow dst$								
Description	The contents of the <i>dst</i> operand are right register's carry bit. This could be viewed is rotated into the MSB of the <i>dst</i> ; at the s the carry bit.	nt-rotated one bit through the status as a 33-bit shift. The carry bit value ame time, the <i>dst</i> LSB is rotated into								
	Rotate right through carry bit:									
		st								
Status Bits	If ST (SETCOND) = 0 and the destinatio flags are modified. If ST (SETCOND) = 1 registers.	n register is R0 – R11, the condition , they are modified for all destination								
	LUFUnaffectedLVUnaffectedUF0NMSB of the outputZ1 if a zero output is generated, 0V0CSet to the value of the bit rotated	otherwise out of the high-order bit								
Mode Bit	OVM operation is not affected by OVM b	bit value.								
Cycles	1									
Example	RORC R4									
	Before Instruction R4 8000 0081h LUF 0 LV 0 UF 1 N 0 Z 0 V 0 C 0	After Instruction R4 4000 0040h LUF 0 LV 0 UF 0 X 0 Z 0 V 0 C 1								

Syntax	RPTB src
Operands	src: 24-bit signed immediate displacement or register mode
Opcode	
	For 24-bit signed immediate or register mode:
	<u>31 24 23 1615 8 7 0</u>
	0 1 1 0 0 1 0 0 <i>src</i> (displacement)
	For register mode:
	<u>31 24 23 1615 8 7 0</u>
	0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
Word Fields	None
Operation	$src + PC + 1 \rightarrow RE$ 1 \rightarrow ST (RM) Next PC \rightarrow RS
Description	RPTB allows a block of instructions to be repeated a number of times without any penalty for looping.
	It activates the block repeat mode of updating the PC. The <i>src</i> operand can be a 32-bit register value or a 24-bit signed immediate value (displacement). The resulting <i>src</i> address is the end address of the block to be repeated. This address is loaded into the repeat end address (RE) register. A 1 is written into the repeat mode bit of status register [ST(RM)] to indicate that the PC is to be updated in the repeat mode. The address of the next instruction is loaded into the repeat-start address (RS) register.
	RE should be greater than or equal to RS (RE \ge RS). Otherwise, the code does not repeat, even though the RM bit remains set to 1.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	4
Example	None

Syntax	RPTBD	src							
Operands	src: 24	4-bit signed im	mediate di	splacem	ent or reg	ister mo	de		
Opcode									
	For 24-bit signed immediate or register mode:								
	31	24 23		1615		87		0	
	0 1 1 0	0 1 0 1		S	rc (displacen	nent)			
	For register mode:								
	31	24 23		1615		87	7	0	
	0 1 1 1	1 0 0 1 1	0 0 0 0 0	0000	0 0 0 0	0000	000	src	
Word Fields	None								
Operation	if <i>src</i> is an immediate value (displacement) <i>src</i> + PC + 3 \rightarrow RE Else: <i>src</i> \rightarrow RE 1 \rightarrow ST(RM) PC of RPTBD + 4 \rightarrow RS								
Description	RPTBD allows a block of instructions to be repeated a number of times without any penalty for looping and with single-cycle execution of the RPTBD instruc- tion. It activates the block repeat mode of updating the PC. The <i>src</i> operand can be a 32-bit register value or a 24-bit signed immediate value (displace- ment). The resulting <i>src</i> address is loaded into the repeat-end address (RE) register (block-end address). A 1 is written to the status-register repeat mode bit [ST(RM)], indicating the PC is to be updated in the repeat-mode. The ad- dress of the next instruction +3 is loaded into the repeat-start address (RS) register.								
	RE shou not repe	ld be greater th at, even thoug	han or equa h the RM b	al to RS (pit remain	(RE≥RS). ns set to 1	Otherw	ise, the	code will	
	RPTBD are exec are not p fore the instruction	does not flush uted and shoul part of the block RPTBD instru ons after RPTE	the pipelir d not modif that is rep action exect 3D.	ne. The the former of the properties of the prop	hree instru gram flow he RC reg should no	uctions f . These t gister mu t be loa	ollowing hree ins ist be lo ded in	RPTBD structions aded be- the three	
	Interrupt	s are disabled	during the	next thr	ee instruct	tions afte	er RPTI	3D.	

Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedXUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

Syntax	BPTS src						
Operands	are: general addressing modes (G)						
Opcode	sic. general-addressing modes (d)						
Opcode	31 24	23	1615	8 7 0			
	0 0 0 1 0 0 1 1	1 G 1	1 0 1 1	src			
Word Fields							
	•	G	src addressing modes	_			
	-	00	register	_			
		01	direct				
		10	indirect				
	_	11	immediate				
Operation	$\begin{array}{l} src \rightarrow \text{RC} \\ 1 \rightarrow \text{ST} (\text{RM}) \\ 1 \rightarrow \text{S} \\ \text{Next PC} \rightarrow \text{RS} \\ \text{Next PC} \rightarrow \text{RE} \end{array}$						
Description	The RPTS instruction times without any po- struction register (IF	on allows a enalty for l R), thus av	a single instruction to be r ooping. Fetches also can oiding repeated memory	repeated a number of be made from the in- access.			
	The <i>src</i> operand is I repeat mode (RM) b peat single bit (S). T only from the instru address (RE) regist	loaded into bit of the sta his indicate ction regis ter and the	the repeat counter (RC). atus register (ST). A 1 also es that the program fetche ter. The next PC is loade repeat-start address (RS	A 1 is written into the o is written into the re- es are to be performed of into the repeat-end S) register.			
	For the immediate n ger and is not sign-	node, the s extended.	<i>src</i> operand is assumed to	be an unsigned inte-			
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected						
Mode Bit	OVM operation is n	ot affected	by OVM bit value.				
Cycles	4						

14-226



The RPTS instruction is not interruptable. Interrupts are held pending until the RPTS instruction is finished executing. In timing-critical applications, this could cause timings to be inaccurate; thus, in timing-critical applications, use caution when using the RPTS instruction.

Syntax	RSQF	F src, dst				
Operands	<i>src:</i> extended-precision register, direct-, and indirect addressing modes <i>dst:</i> extended-precision register					
Opcode						
	31		24 23	1615	8 7	0
	0 0 0	1 1 1 0	01G	dst	src	
Word Fields						
			G	src addressing	modes	
			00	extended-precis	ion register	
			01	direct		
			10	indirect		
			11	16-bit immediate	9	
Operation	16-bit	reciprocal	of the squa	are root of $src ightarrow c$	lst	
Description	The 10 the <i>sro</i> is assi	6-bit approx coperand is umed to be	ximation of s loaded in e positive. 7	the reciprocal of tl to the <i>dst</i> register. The operation for r	he square root of the n The number at the <i>src</i> negative inputs is unde	umber at operand efined.
	The va bers.	alue at the	dst and src	operands are ass	sumed to be floating-po	oint num-
Status Bits	LUF LV UF N Z V C	Unchange 1 if input is 0 0 1 if input is Unaffected	ed s zero uncl s zero, 0 of d	nanged otherwise herwise		
Mode Bit	OVM	operation is	s not affect	ed by OVM bit val	ue.	
Cycles	1					
Example	None					

Syntax	SIGI src, dst					
Operands	<i>src:</i> direct- and i <i>dst:</i> register mo	ndirect-a de (assu	addressin med to be	g modes (assumed e signed integer)	to be signed inte	ger)
Opcode						
	31 24	1 23	16	15	8 7	0
	0 0 0 1 0 1 1 0	0 G	dst		SIC	
Word Fields						
		G	src ad	ddressing modes	_	
		01	direct			
		10	indire	ct	_	
Operation	LOCK (or LLOCK)	pin brou	ight low			
	$\frac{src \rightarrow dst}{LOCK} \text{ (or } \overline{LLOCK})$	pin brou	ıght high			
Description	An interlocking ope or LLOCK) if, and and <i>dst</i> operands formed, the bus-loc formed, SIGI perfo to Section 9.7 on p	ration is only if, a are assu k signal rms the page 9-3	signaled In externation Imed to b is deasse read but 9 for a de	by the appropriate t al-memory access be signed integers. erted. If an internal-r does not assert a t tailed description.	bus-lock signal (LC is performed. The After the read is p nemory access is p ous-lock signal. R	DCK e <i>src</i> per- per- efer
	The numbers at th	e <i>src</i> and	d <i>dst</i> ope	rands are treated a	as signed integers	
Status Bits	If ST (SETCOND) flags are modified. registers.	= 0 and If ST (SI	the destir ETCOND	nation register is R() = 1, they are mod	0 – R11, the condi ified for all destina	ition Ition
	LUFUnaffectedLVUnaffectedUF0N1 if a negatiZ1 if a zero reV0CUnaffected	ve resuli esult is g	t is generated	ated, 0 otherwise , 0 otherwise		
Mode Bit	OVM operation is	not affec	ted by O	/M bit value.		
Cycles	1					
Example	None					

Syntax	STF src, dst					
Operands	<i>src</i> : register (R0 – R11) <i>dst:</i> general-addressing modes (G)					
Opcode						
	31 24	1 23	1615	8 7 0		
	0 0 0 1 0 1 0 0	0 G	src	dst		
Word Fields						
			are addressing modes	_		
		01	direct	_		
		10	indirect			
				_		
Operation	$src \rightarrow dst$					
Description	The <i>src</i> register is I ands are assumed	oaded into to be floati	the <i>dst</i> memory location. ⁻ ng-point numbers.	The <i>src</i> and <i>dst</i> oper-		
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected					
Mode Bit	OVM operation is r	not affected	l by OVM bit value.			
Cycles	1					

Example

STF R2,@98A1h

	Before Instruction
DP	80h
R2	052 C501 900h 4.30782204e + 01
Data	at 80 98A1h
	0h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

	After Instruction
DP	80h
R2	052 C501 900h 4.30782204e + 01
Data	at 80 98A1h
	052 C5019h 4.30782204e + 01
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

Syntax	STFI sra	c, dst				
Operands	<i>src</i> : register (R0 – R11) <i>dst</i> : general-addressing modes (G)					
Opcode						
	31	24	23	1615	8 7 0	
	0 0 0 1	0 1 0 0	1 G	src	dst	
Word Fields						
		-	G	src addressing modes		
			01	direct		
		-	10	Indirect		
Operation Description	$src \rightarrow ds$ Signal e The src	<i>st</i> nd of interlo register is lo	ocked ope	ration. the <i>dst</i> memory location	n. An interlocked opera-	
	tion is sig to be flo informat	gnaled over ating-point ion.	numbers.	LOCK. The src and dsi Refer to Section 9.7 or	n page 9-39 for detailed	
Status Bits	LUF U LV U UF U N U Z U V U C U	Inaffected Inaffected Inaffected Inaffected Inaffected Inaffected				
Mode Bit	OVM op	eration is n	ot affected	l by OVM bit value.		
Cycles	1					

Example	STFI R3,*-AR4			
	Before Inst	truction	After Instru	ction
	R3 07 33C0 0000h	1.79750e + 02 R	3 07 33C0 0000h	1.79750e + 02
	AR4 80 993Ch	A	.R4 80 993Ch	
	Data at 80 993Bh	D	Pata at 80 993Bh 733 C000h	1.79750e + 02
	LUF 0	L	UF 0	
	LV 0	Ľ	V 0	
	UF 0	U	IF 0	
	N 0	Ν	0	
	Z 0	Z	0	
	V 0	V	0	
	C 0	C	0	

STF||STF Parallel Store Floating-Point Value

Syntax	STF src2, o	dst2 dst1						
Operands	src1; register (Rn1, $0 \le n1 \le 7$) dst1: indirect (disp = 0, 1, IR0, IR1) src2: register (Rn2, $0 \le n2 \le 7$) dst2: indirect (disp = 0, 1, IR0, IR1)							
Opcode								
	31	24 23		1615	8	7	0	
	1 1 0 0 0 0	0 <i>src2</i>	0 0 0	src1	dst1	dst2		
Word Fields	None							
Operation	$src2 \rightarrow dst2$ $src1 \rightarrow dst2$	2						
Description	Two STF instrute to be floating-p	ictions are point numb	executed ers.	in parallel.	Both <i>src1</i> and s	src2 are assum	ned	
Status Bits	LUF Unaffect LV Unaffect UF Unaffect N Unaffect Z Unaffect V Unaffect C Unaffect	ted ted ted ted ted ted ted						
Mode Bit	OVM operation	n is not affe	ected by	OVM bit va	lue.			
Cycles	1							
Example	STF R4	l,*AR3						
	STF R3	3,*++AR5	lia m		A fa a su la s			
	B4 07.0C80.0		(ION 4050e ± 02	P R4	After Instruction			
	AR3 80 9	835h	.40000 1 02	AR3	80 9834h]	02	
	R3 07 33C0 0	 000h 1.7	79750e + 02	2 R3	07 33C0 0000h	1.79750e +	02	
	AR5 80 9	9D2h		AR5	5 80 99D3h]		
	Data <u>at 80 9835h</u>			Data	a <u>at 80 9835h</u>	1		
		<u>0h</u>]		Det	070C 8000h] 1.4050e +	02	
	Data <u>at 80 99D3</u>	0h		Data	0733 C00000h] 1.79750e +	02	
	LUF	0		LUF	0]		
	LV	0		LV	0]		
	UF	0		UF	0]		
	N	0		Ν	0]		
	Z	0		Z	0]		
	V	0		V	0]		
	С	0		С	0	J		

Syntax	STI src, dst		
Operands	<i>src:</i> register (any r <i>dst:</i> general-addre	egister in CPU prin ssing modes (G)	nary-register file)
Opcode	<u>31 24 2</u>	3 <u>1615</u>	8 7 0
Word Fields		G	usi
	_	G <i>src</i> addre	essing modes
	_	01 direct	
	_	10 indirect	
Operation	src ightarrow dst		
Description	The <i>src</i> register is loa ands are assumed to	aded into the <i>dst</i> me	emory location. The <i>src</i> and <i>dst</i> oper- s.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected		
Mode Bit	OVM operation is not	t affected by OVM	bit value.
Cycles	1		
Example	STI R4, @982Bh Before Ins DP 80h R4 42BD7h Data at 80 982Bh 0E5FCh LUF 0 LV 0 UF 0 X 0 Z 0 V 0 C 0	truction 273 367 58 876	After Instruction DP 80h R4 42BD7h 273 367 Data 480 982Bh 273 367 LUF 0 273 367 LUF 0 273 367 LV 0 1 Q 0 1 LV 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 1 Q 0 2 Q 0 2 Q 0 2 Q 0 2 Q 0 2 Q 0

Syntax	STII src, dst		
Operands	<i>src</i> : register (any i <i>dst</i> : general-addre	egister in CPU prin essing modes (G)	nary-register file)
Opcode			
	31 24 2	3 1615	8 7 0
	0 0 0 1 0 1 0 1 1	G src	dst
Word Fields			
	-	G src addre	essing modes
	-	01 direct	
		10 indirect	
	_		
Operation	$src \rightarrow dst$ Signal end of interloo	cked operation.	
Description	The <i>src</i> register is loat tion is signaled over \overline{I} to be signed integers tion.	aded into the <i>dst</i> me OCK or LLOCK. Th Refer to Section S	emory location. An interlocked opera- he <i>src</i> and <i>dst</i> operands are assumed 9.7 on page 9-39 for detailed informa-
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected		
Mode Bit	OVM operation is no	t affected by OVM	bit value.
Cycles	1		
Example	STII R1,098AEh		
	Before Ins	struction	After Instruction
	DP 80h		DP 80h
	R1 78Dh		R1 78Dh
	Data <u>at 80 98AEh</u> 25Ch		Data <u>at 80 98AEh</u> 78Dh

Syntax	STI <i>src2, ds</i> STI <i>src1, ds</i>	t2 t1				
Operands	src1: register (l dst1: indirect (c src2: register (l dst2: indirect (c	Rn1, 0 ≤ disp = 0, Rn2, 0 ≤ disp = 0,	n1 ≤ 7) 1, IR0, IF n2 ≤ 7) 1, IR0, IF	R1) R1)		
Opcode						
	31	24 23		16	15 8	<u> </u>
	1 1 0 0 0 1	src2	000	src1	dst1	dst2
Word Fields	None					
Operation	$src2 \rightarrow dst2$ $src1 \rightarrow dst1$					
Description	Two integer store same address, t	es are pe he value	rformed written i	in paral s that o	lel. If both stores f STI <i>src2, dst2.</i>	are executed to the
Status Bits	LUF Unaffecte LV Unaffecte UF Unaffecte N Unaffecte Z Unaffecte V Unaffecte C Unaffecte	ed ed ed ed ed ed				
Mode Bit	OVM operation i	s not affe	ected by	OVM b	it value.	
Cycles	1					

STI||**STI** Parallel STI and STI

Example

STI R0,*++AR2(IR0) ||STI R5,*AR0

1101	110, 1110						
	Before In	struction		After Instruction			
R0	0DCh	220	R0 🗌	0DCh	220		
AR2	80 9830h		AR2	80 9838h			
IR0	8h		IR0	8h			
R5	35h	53	R5	35h	53		
AR0	80 98D3h		AR0	80 98D3h			
Data	at 80 9838h 0h		Data at	t 80 9838h 0DCh	220		
Data	at 80 98D3h		Data <u>at</u>	t 80 98D3h			
	0h			35h	53		
LUF	0		LUF	0			
LV	0		LV	0			
UF	0		UF 🗌	0			
N	0		N	0			
Z	0		z 🗌	0			
V	0		V	0			
С	0		сГ	0			

14-238

Syntax	STIK src, dst
Operands	<i>src;</i> 5-bit signed integer <i>dst:</i> direct and indirect mode
Opcode	<u>31 24 23 1615 8 7 0</u>
	0 0 0 1 0 1 0 1 0 G <i>src</i> dst
Word Fields	
	G src addressing modes
	00 direct
	11 indirect
Operation	$src \rightarrow dst$
Description	The 5-bit signed integer <i>src</i> value is loaded into the <i>dst</i> memory location. The <i>src</i> and dst operands are assumed to be signed integers.
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

Syntax	SUBB src	, dst					
Operands	<i>src:</i> ger <i>dst:</i> reg	neral-addres ister (any re	ssing mo egister ir	odes (G) I CPU primary-	register file)		
Opcode							
	31	24 23	3	1615	8	3 7	0
	0 0 0 1	0 1 1 0 1	G	dst	5	src	
Word Fields							
			G	<i>src</i> addressing	g modes	-	
			00	register (any re CPU primary-re	egister in egister file)		
			01	direct			
			10	indirect			
			11	immediate		_	
Operation	dst – src -	$- C \rightarrow dst$					
Description	The different into the <i>ds</i> gers.	ence of the o at register. T	<i>dst, src, a</i> he <i>dst</i> ar	and C operands nd <i>src</i> operands	s, as calculat s are assume	ted above, is load ed to be signed int	ed te-
Status Bits	If ST (SET flags are r registers.	COND) = 0 nodified. If 8) and the ST (SET)	e destination re COND) = 1, the	gister is R0 - ey are modifie	- R11, the condition ad for all destination	on on
	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C 1 if a borrow occurs, 0 otherwise 						
Mode Bit	OVM oper	ration is affe	ected by	OVM bit value			
Cycles	1						

SUBB *AR5++(4),	R5				
Before Ins	After In	After Instruction			
AR5 80 9800h		AR5 80 9804	h		
R5 0FAh	250	R5 032	h 50		
Data at 80 9800h		Data at 80 9800h			
0C7h	199	0C7	h 100		
LUF 0		LUF	0		
LV 0		LV	0		
UF 0		UF	0		
N 0		Ν	0		
Z 0		Z	0		
V 0		V	0		
C 1		C	0		

Example

Syntax	SUBB3 src2, src1, dst							
Operands	<i>src1, src2</i> : type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)							
Opcode								
	Туре	e 1						
	31	1 24 23 161			161	5 87 0		
	0 0	1 0 0 1 1	0 0	Т	dst	src1	src2	
	Туре	e 2						
	31		24 23		161	5	87 0	
	0 0	1 1 0 1 1	0 0	Т	dst	src1	src2	
Word Fields								
True 4								
Туре 1			-					
		src1 addre	ssing n	nodes		src2 addressin	g modes	
	00	register mode (any CPU register)			register)	register mode (any CPU register)		
	01	indirect mode (disp = 0, 1, IR0, IR1)			, IR0, IR1)	register mode (any CPU register)		
	10	register mode (any CPU register)			register)	indirect mode (disp = 0, 1, IR0, IR1)		
	11	indirect mod	de (disp	0 = 0, 1	, IR0, IR1)	indirect mode (d	isp = 0, 1, IR0, IR1)	
Туре 2								
	Т	src1 addre	ssing n	nodes	;	src2 addressin	g modes	
	00	register mo	de (any	CPU	register)	8-bit signed imm	rediate	
	01	register mo	de (any	CPU	register)	indirect mode *+ displacement)	ARn(5-bit unsigned	
	10	indirect mode *+ARn(5-bit unsigned displacement)			it unsigned	8-bit signed immediate		
	11	indirect mod displaceme	de *+AF nt)	Rn1(5-I	bit unsigned	indirect mode *+ displacement)	ARn2(5-bit unsigned	
Operation	src1	– <i>src2</i> – C	\rightarrow dst					

Description	The difference of the <i>src1</i> and <i>src2</i> operands and the C (carry) flag is loaded into the <i>dst</i> register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be signed integers.			
Status Bits	If ST (SETCOND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SETCOND) = 1, they are modified for all destination registers.			
	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C 1 if a borrow is generated, 0 otherwise 			
Mode Bit	OVM operation is affected by OVM bit value.			
Cycles	1			
Example	None			
SUBC Subtract Integer Conditionally

Syntax	SUBC	src, dst						
Operands	<i>src</i> : general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)							
Opcode								
	31	:	24 23	1615	5	8 7	0	
	0 0 0	1011	10G	dst		SIC		
Word Fields								
			G	src add	Iressing modes	—		
			00	register CPU pr	(any register in imary-register file)	—		
			01	direct				
			10	indirect				
			11	immedi	ate			
Description	(<i>dst</i> – Else: <i>dst</i> << The <i>sr</i> with a greate nifican is less <i>dst</i> and	src << 1) C $1 \rightarrow dst$ coperand is value that r than or eq t bit is set to than zero, a d <i>src</i> opera	DR 1 \rightarrow dst s subtracted depends up ual to zero, o 1, and the dst is left-sh unds are ass	d from the con the re then (<i>dst</i> - result is ifted one sumed to	<i>dst</i> operand. The esult of the subtra – <i>src)</i> is left-shifted loaded into the <i>ds</i> bit and loaded int be unsigned inte	<i>dst</i> operand is lo action. If (<i>dst</i> – s done bit, the leas <i>t</i> register. If (<i>dst</i> o the <i>dst</i> registe gers.	baded src) is st-sig- - src) r. The	
	SUBC	can be use	ed to perfor	m a singl	e step of a multib	it-integer divisio	n.	
Status Bits	LUF LV UF N Z V C	Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected Unaffected	9 9 9 9 9 9 9					
Mode Bit	OVM o	peration is	not affecte	d by OVI	V bit value.			
Cycles	1							
14-244								

Example 1 SUBC @98C5h, R1	
Before Instruction After Inst	ruction
DP 80h DP 80h	
R1 04F6h 1270 R1 0C9h	201
Data at 80 98C5h Data at 80 98C5h Data at 80 98C5h	
492h 1170 492h	1170
LV 0 LV 0	
UF 0 UF 0	
N 0 N 0	
Z 0 Z 0	
V 0 V 0	
C 0 C 0	
Example 2 SUBC 3000, R0 (3000 = 0BB8h)	
Before Instruction After Instr	ruction
R0 07D0h 2000 R0 0FA0h	4000
LV 0 LV 0	
UF 0 UF 0	
N 0 N 0	
V 0 V 0	

Syntax	SUBF	src, dst
Operands	src: dst:	general-addressing modes (G) register (R0 – R11)
Opcode		

31	24 23			3	16	15 8 7	0		
000	10	1	1	1	1	G	dst	src	

Word Fields

G	src addressing modes	
00	register (R0-R11)	
01	direct	
10	indirect	
11	immediate	

Operation	$dst - src \rightarrow dst$
Description	The result of <i>the dst</i> operand minus the <i>src</i> operand is loaded into the <i>dst</i> regis- ter. The <i>dst</i> and <i>src</i> operands are assumed to be floating-point numbers.
Status Bits	 LUF 1 if a floating-point underflow occurs, unchanged otherwise LV 1 if an floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an floating-point overflow occurs, 0 otherwise C Unaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1

SU	ВF	*AR0(TR	0).R5
50		Before li	nstruction
AF	80	80 9888h	
IR) [80h	
R5		07 33C0 0000h	1.79750000e + 02
Da	ta <u>a</u>	t 80 9888h	
	L	70C 8000h	1.4050e + 02
LU	F	0	
LV		0	
UF		0	
Ν		0	
Z		0	
V		0	
С	Г	0	

After Instruction

80 9808h 80h R5 05 1D00 0000h 3.9250e + 01 Data at 80 9888h 70C 8000h 1.4050e + 02

LUF	0	
LV	0	
UF	0	
Ν	0	

AR0

IR0

0 Ζ

0 ۷

0 С

Syntax	SUE	F3 src	2, src	1, dsi	t			
Operands	<i>src1, src2</i> :type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (R0 – R11)							
Opcode								
	Туре	e 1						
	31			24 23		161	5 8	3 7 0
	0 0	1 0 0	11	01	Т	dst	src1	src2
	Туре	e 2						
	31			24 23		161	5 8	3 7 0
	0 0	1 1 0	1 1	01	Т	dst	src1	src2
Word Fields Type 1								
	Т	src1 a	ddres	sing	mode	S	src2 addressing	modes
	00	registe	er mod	e (R0	–R11)		register mode (R	0–R11)
	01	indired	t mode	e (disp	p = 0,	1, IR0, IR1)	register mode (R	0–R11)
	10	registe	r mod	e (R0·	–R11)		indirect mode (dis	sp = 0, 1, IR0, IR1)
	11	indired	t mode	e (disp	p = 0,	1, IR0, IR1)	indirect mode (dis	sp = 0, 1, IR0, IR1)
Туре 2	_							
	Т	src1 a	ddres	sing r	mode	S	src2 addressing	modes
	01	registe	r mode	e (any	/ CPU	register)	indirect mode *+A displacement)	ARn(5-bit unsigned
	11	indirec displac	t mode ement	e *+AF t)	Rn1(5∙	bit unsigned	indirect mode *+A displacement)	ARn2(5-bit unsigned
Operation	src1	– src2	ightarrow ds	t				

Description	The difference of the <i>src1</i> and <i>src2</i> operands is loaded into the <i>dst</i> register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be floating-point numbers.
Status Bits	 LUF 1 if a floating-point underflow occurs, unchanged otherwise LV 1 if an floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an floating-point overflow occurs, 0 otherwise C Unaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1
Example	None

Syntax	SUBF3 STF	src1, src2, o src3, dst2	dst1				
Operands	src1: regi src2: indin dst1: regi src3: regi dst2: indin	ster (R0 – R7 rect (disp = 0, ster (R0 – R7 ster (R0 – R7 rect (disp = 0,	′) , 1, IR0, ′) , 1, IR0,	IR1) IR1)			
Opcode							
	31	24 23	0101	16	15 E	3 7 0	
			SICI	SIC3	0512	SIC2	
Word Fields	None						
Operation	src2-s $src3 \rightarrow$	src1 ightarrow dst1					
Description	A floating-point subtraction and a floating-point store are performed in parallel. All registers are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STF) reads from a reg- ister and the operation being performed in parallel (SUBF3) writes to the same register, then STF accepts as input the contents of the register before it is mo- dified by the SUBF3						
	If <i>src3</i> and	<i>dst1</i> point to t	he same	locatio	on, <i>src3</i> is read befo	re the write to dst1.	
Status Bits	 LUF 1 if a floating-point underflow occurs, unchanged otherwise LV 1 if an floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an floating-point overflow occurs, 0 otherwise C Unaffected 						
Mode Bit	OVM opera	ation is not af	fected b	y OVM	l bit value.		
Cycles	1						

Exa	m	pl	e

SUBF3 R1,*-AR4(IR1),R0 || STF R7,*+AR5(IR0)

		11(0 (11(0))
	Before In	struction
R1	05 7B40 0000h	6.28125e + 01
AR4	80 98B8h	
IR1	8h	
R0	0h	
R7	07 33C0 0000h	1.79750e + 02
AR5	80 9850h	
IR0	10h	
Data	at 80 98B0h 70C 8000h	1.4050e + 02
Data	at 80 9860h 0h	
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

	After Inst	truction
R1	05 7B40 0000h	6.28125e + 01
AR4	80 98B8h	
IR1	8h	
R0	06 1B60 0000h	7.768750e + 01
R7	07 33C0 0000h	1.79750e + 02
AR5	80 9850h	
IR0	10h	
Data	at 80 98B0h	
	70C 8000h	1.4050e + 02
Data	at 80 9860h	
	733 C000h	1.79750e + 02
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

Syntax	SUBI	src, dst					
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)						
Opcode							
	31	24	23	16	15	8 7 (0
	000	11000	0 G	ast		src	
Word Fields							
						-	
			G	src a	ddressing modes	_	
			00	regist CPU	er (any register in primary-register file)		
			01	direct			
			10	indire	ct		
			11	imme	diate	_	
Operation	dst – s	$src \rightarrow dst$					
Description	The reater. Th	sult of the <i>dst</i> e <i>dst</i> and <i>src</i>	operand n operands	ninus th are as	ne <i>src</i> operand is load ssumed to be signed	ded into the <i>dst</i> regis d integers.	3-
Status Bits	If ST (flags a registe	SETCOND) = re modified. ers.	= 0 and the If ST (SET	e destir COND	nation register is R0) = 1, they are modif	 R11, the conditio ied for all destinatio 	n n
	LUF LV UF	Unaffected 1 if an intege	er overflow	occur	s, unchanged other	wise	
	N Z	1 if a negative 1 if a zero re	ve result is esult is aer	gener Ierated	ated, 0 otherwise		
	V	1 if an intege	er overflow	occur	s, 0 otherwise		
	C	1 if a borrow	occurs, 0	otherv	VISE		
Mode Bit	OVM o	peration is a	affected by	OVM	bit value.		
Cycles	1						

SUBI 220,R7

550

R7	226h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

After Instruction

330

14Ah R7 0 LUF 0 LV 0 UF Ν 0 0 Ζ 0 ٧ С 0

SUBI3 Subtract Integer, 3 Operands

Syntax	SUE	813 src2, src1	, dst					
Operands	src1, src2:type 1 or type 2 three-operand addressing modes							
	dst: register mode (any register in CPU primary-register file)							
Oncode		-						
opcouc								
	Туре 1							
	31		24 23	161	5 8	3 7 0		
	0 0	1 0 0 1 1	1 0 T	dst	src1	src2		
	Тур	e 2						
	31		24 23	161	5 8	3 7 0		
	0 0	1 1 0 1 1	1 0 T	dst	src1	src2		
Word Fields								
Turne 4								
турет	. .			_				
	<u> </u>	src1 address		S	sicz audressing modes			
	00	indirect mode (diar _ 0, 1, 100, 101)		register mode (an	iy CPU register)			
	10	indirect mode (disp = 0, 1, IRU, IRT)		indiract mode (di				
	10	indirect mode (dian 0.1 ID0 ID1)		indirect mode (dis	sp = 0, 1, IR0, IR1)			
			e (uisp = 0,	וחו, וחו)		p = 0, 1, 10, 11)		
Туре 2								
	Т	src1 address	sing mode	S	src2 addressing	modes		
	00	register mode	e (any CPL	J register)	8-bit signed imme	ediate		
	01 register mode (any CPU register) indirect mode *+ARn(5-bit unsigned displacement)				ARn(5-bit unsigned			
	10	indirect mode *+ARn(5-bit unsigned displacement)		8-bit signed immediate				
	11	indirect mode displacement	e *+ARn1(5 :)	i-bit unsigned	indirect mode *+A displacement)	ARn2(5-bit unsigned		
Operation	src1	$- src2 \rightarrow ds$	t					

Description	The result of the <i>src1</i> operand minus the <i>src2</i> operand is loaded into the <i>dst</i> register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be signed integers.			
Status Bits	If ST (SETCOND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SETCOND) = 1, they are modified for all destination registers.			
	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C 1 if a borrow is generated, 0 otherwise 			
Mode Bit	OVM operation is affected by OVM bit value.			
Cycles	1			
Example	None			

Syntax	SUBI3 STI	src1, src2, ds src3, dst2	t1			
Operands	src1:register (R0 - R7)src2:indirect (disp = 0, 1, IR0, IR1) $dst1:$ register (R0 - R7)src3:register (R0 - R7) $dst2:$ indirect (disp = 0, 1, IR0, IR1)					
Opcode						
	31	24 23		16	15 8	7 0
	1 1 1 0 1	1 0 <i>asti</i>	SIC 1	SIC3	ast2	SIC2
Word Fields	None					
Operation	src2 - s $src3 \rightarrow$	src1 ightarrow dst1				
Description	An integer subtraction and an integer store are performed in parallel. All regis- ters are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (SUBI3) writes to the same register, then STI accepts as input the contents of the register before it is modified by the SUBI3.					
	If <i>src3</i> and o	<i>dst1</i> point to the	same loc	ation, <i>sr</i>	c3 is read befor	e the write to dst1.
Status Bits	 LUF Unaffected LV 1 if an integer overflow occurs, unchanged otherwise UF 0 N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if an integer overflow occurs, 0 otherwise C 1 if a borrow occurs, 0 otherwise 					
Mode Bit	OVM operation is affected by OVM bit value.					
Cycles	1					

SUBI3 R7,*+AR2(IR0),R1 || STI R3,*++AR7

511 RS, *+-	FAR /		
Before In	struction	Afte	r Instruction
14h	20	R7	14h 20
80 982Fh		AR2 80 98	2Fh
10h		IR0	10h
0h		R1 00	C8h 200
35h	53	R3	35h 53
80 983Bh		AR7 80 983	3Ch
at 80 983Fh		Data <u>at 80 983Fh</u>	
0DCh	220	0	<u>Ch</u> 220
at 80 983Ch		Data <u>at 80 983Ch</u>	
0h			35h 53
0		LUF	0
0		LV	0
0		UF	0
0		N	0
0		Z	0
0		V	0
0		С	0
	Before Ir 14h 80 982Fh 10h 0h 35h 80 983Fh 0DCh at 80 983Ch 0h 0	Before Instruction 14h 20 80 982Fh 10h 0h 35h 53 80 983Fh 53 80 983Fh 220 at 80 983Fh 220 at 80 983Ch 0h 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Before Instruction Afte 14h 20 R7 80 982Fh AR2 80 982 10h IR0 100 0h R1 00 35h 53 R3 100 35h 53 R3 100 10h R1 00 00 10h 220 01 01 10h 220 01 01 100 220 01 01 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100

Syntax	SUBRB src, dst			
Operands	<i>src:</i> general-addressing modes (G) <i>dst:</i> register (any register in CPU primary-register file)			
Opcode				
	31 24 2	23	1615	870
	0 0 0 1 1 0 0 0 1	G	dst	src
Word Fields				
	_			_
	_	G	src addressing modes	_
		00	register (any register in CPU primary-register file)	
		01	direct	
		10	indirect	
	_	11	immediate	_
Operation	$\textit{src}-\textit{dst}-C \rightarrow \textit{dst}$			
Description	The difference of the into the <i>dst</i> register. gers.	e <i>src, dst, a</i> The <i>dst</i> ar	and C operands, as calcula nd <i>src</i> operands are assum	ated above, is loaded ned to be signed inte-
Status Bits	If ST (SETCOND) = flags are modified. If registers.	0 and the ST (SET)	destination register is R0 COND) = 1, they are modif	- R11, the condition ied for all destination
	LUFUnaffectedLV1 if an integerUF0N1 if a negativeZ1 if a zero resV1 if an integerC1 if a borrow	r overflow e result is sult is gen r overflow occurs, 0	occurs, unchanged othen generated, 0 otherwise erated, 0 otherwise occurs, 0 otherwise otherwise	wise
Mode Bit	OVM operation is af	fected by	OVM bit value.	
Cycles	1			

SUBRB R4,R6

	Before Instruction
R4	03CBh
R6	0258h
LUF	0
LV	0
UF	0
N	0
Z	0
V	0
С	1

After Instruction	
03CBh	971
0172h	370
0	
0	
0	

N 0 Z 0

R4

R6

LUF LV UF

971

600

v _____0

C 0

Syntax	SUBRF src, dst		
Operands	src: dst:	general-addressing modes (G) register (R0 – R11)	
Opcode			

31		24 23	3	16	15 8 7	0
000	11	0 0 1 0	G	dst	src	

Word Fields

G	src addressing modes	
00	register (R0-R11)	
01	direct	
10	indirect	
11	immediate	

Operation	$src - dst \rightarrow dst$
Description	The result of the <i>src</i> operand minus the <i>dst</i> operand is loaded into the <i>dst</i> regis- ter.The <i>dst</i> and <i>src</i> operands are assumed to be floating-point numbers.
Status Bits	 LUF 1 if a floating-point underflow occurs, unchanged otherwise LV 1 if a floating-point overflow occurs, unchanged otherwise UF 1 if a floating-point underflow occurs, 0 otherwise N 1 if a negative result is generated, 0 otherwise Z 1 if a zero result is generated, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise V 1 if a floating-point overflow occurs, 0 otherwise C Unaffected
Mode Bit	OVM operation is not affected by OVM bit value.
Cycles	1

Subtract Reverse Floating-Point Values SUBRF

Example

SUBRF @9905h,R5

	Before In	struction
DP	80h	
R5	05 7B40 0000h	6.281250e + 01
Data	at 80 9905h	
	733 C000h	1.79750e + 02
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

	A file v luce	A
	After Ins	truction
DP	80h	
R5	06 69E0 0000h	1.16937500e + 02
Data	at 80 9905h	
	733 C000h	1.79750e + 02
LUF	0	
LV	0	
UF	0	
Ν	0	
Ζ	0	
V	0	
С	0	

SUBRI Subtract Reverse Integer

Syntax	SUBRI src, dst			
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)			
Opcode	31 24	23	1615	870
		1 G	dst	src
Word Fields				
				_
		00	register (any register in CPU primary-register file)	_
		01	direct	
		10	indirect	
		11	immediate	
Operation	$src - dst \rightarrow dst$	operand m	inus the <i>d</i> stoperand is load	ded into the <i>dst</i> regis-
Description	ter. The <i>dst</i> and <i>src</i>	operands	are assumed to be signed	d integers.
Status Bits	If ST (SETCOND) = flags are modified. I registers.	= 0 and the If ST (SET(destination register is R0 COND) = 1, they are modif	 R11, the condition ied for all destination
	LUFUnaffectedLV1 if an integerUF0N1 if a negativeZ1 if a zero redV1 if an integerC1 if a borrow	er overflow ve result is esult is gen er overflow v occurs. 0	occurs, unchanged otherv generated, 0 otherwise erated, 0 otherwise occurs, 0 otherwise otherwise	wise
Mode Bit	OVM operation is a	affected by	OVM bit value.	
Cycles	1			

SUBRI	*AR5++(IR0),R3
	Before Instruction
AR5	80 9900h
IR0	8h
R3	0DCh
Data at a	80 9900h 226h
LUF	0
LV	0
UF	0
N	0
Z	0
V	0
с 🗌	0

After Instruction

226h

0

0 0

0

0

0

AR5

IR0

R3 [

LUF

LV UF

Ν

Ζ

v c

Data at 80 9900h

220

550

80 9908h 8h 014Ah

330

550

Syntax	SWI		
Operands	None		
Opcode			
	31 24 23 1615 8 7 0		
	0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0		
Word Fields	None		
Operation	Performs an emulation interrupt		
Description	The SWI instruction performs an emulator interrupt. This is a reserved instruc- tion and should not be used in normal programming.		
Status Bits	LUFUnaffectedLVUnaffectedUFUnaffectedNUnaffectedZUnaffectedVUnaffectedCUnaffected		
Mode Bit	OVM operation is not affected by OVM bit value.		
Cycles	4		
Example	None		

Syntax	TOIEE	EE src, dst
Operands	src: dst:	extended-precision register (R0 – R11), direct- and indirect-addressing modes extended-precision register
Opcode		

31						24	23		16	15	8 7	0
0 0	0	1	1	0	1	1	1	G	dst		src	

Word Fields

			G	<i>src</i> addressing modes	
			00	register [extended-precision register (R0-R11)]	
			01	direct	
			10	indirect	
			11	immediate	_
Operation	conve	ert <i>src</i> to IEEE	E format -	\rightarrow dst	
Description	The s the IE	<i>src</i> operand is EEE floating-p	converte oint form	d from a 2s-complement floa at.	ting-point format to
	The s cept f mat. ⁻ be us	crc operand is a or the immedia The converted and to store th	assumed ate mode d result go e result to	to be a single-precision floatin that is considered a short 16- bes into the 32 MSBs of the <i>d</i> o memory.	ig-point number, ex- bit floating point for- <i>st</i> register. STF can
Status Bits	LUF LV UF N Z V C	Unaffected 1 if an overf 0 1 if a negativ 1 if a zero re 1 if an overf Unaffected	low occur ve result esult is ge low occur	s, unchanged otherwise s generated, 0 otherwise nerated, 0 otherwise s, 0 otherwise	
Mode Bit	OVM	operation is r	not affecte	ed by OVM bit value.	
Cycles	1				
Example	None				

TOIEEE||**STF** Parallel TOIEEE and STF

Syntax	тс ST	DIEEE &	src2, d src3, d	lst1 lst2						
Operands	src2: dst1: src3: dst2:	indirect register register indirect	mode mode mode mode	(disp e (Rn1 e (Rn1 e (disp	= 0, 1, IF , 0 ≤ n1 ≤ , 0 ≤ n1 ≤ = 0, 1, IF	R0, IR1) ≨ 7) ≨ 7) R0, IR1)				
Opcode										
	31 1 1 1	100	24 2 0 0	23 dst1	0 0 0	16 src3	015 dst2	8 7	src2	0
Word Fields	None									
Operation	conver in para <i>src3</i> →	rt <i>src2</i> to Illel with <i>dst2</i>	IEEE	forma	at $\rightarrow dst$,				
Description	The <i>sr</i> the IE	<i>c</i> 2 opera EE floati	and is ng-poi	conve int fori	erted from mat.	a 2s-co	mplement	floatin	g-point for	mat to
	The <i>sr</i> The co point s	c2 oper onverted tore is c	and is I resul [:] Ione ir	assur t goes 1 para	ned to be into the llel.	a single 32 MSE	e-precision 3s of the <i>d</i>	n floatir <i>st1</i> reg	ng-point nu gister. A flo	mber. ating-
	lf <i>src2</i> to <i>dst2</i>	and <i>dsi</i> ?.	2 poin	t to th	e same lo	ocation,	then <i>src2</i> i	is read	before the	write
Status Bits	LUF LV UF N Z V C	Unaffec 1 if an c 0 1 if a ne 1 if a ze 1 if an c Unaffec	eted overflo egative ero res overflo eted	w occ e resul sult is g w occ	urs, unch It is gene generated urs, 0 oth	anged o rated, 0 d, 0 othe nerwise	otherwise otherwise erwise			
Mode Bit	OVM o	operatio	n is no	ot affec	cted by O	VM bit v	value.			
Cycles	1									
Example	None									

Syntax	TRAP <i>cond</i> N					
Operands	N: immediate mode ($0 \le N \le 511$)					
Opcode						
	31 24 23 1615 8 7 0 0 1 1 1 0 1 0 0 0 0 0 cond 0 0 0 0 0 N					
Word Fields	None					
Operation	If (<i>cond</i> is true)					
	$\begin{array}{l} ST(GIE) \rightarrow ST(PGIE) \\ ST(CF) \rightarrow ST(PCF) \\ 0 \rightarrow ST(GIE) \\ 1 \rightarrow ST(CF) \\ next PC \rightarrow ^*(++SP) \\ trap vector N \rightarrow PC \end{array}$					
	Else, continue.					
Description	If the condition is true, then GIE and CF are saved in PGIE and PCF in the status register, all interrups are disabled (0 \rightarrow GIE), and the cache is frozen (1 \rightarrow CF). Then, the contents of the PC are pushed onto the system stack, and the PC is loaded with the contents of the specified trap vector (N). If the condition is not true, then continue normal operation.					
	If traps are to be nested, you may need to save the status register before executing TRAP cond.					
Status Bits	GIESet to 0 if TRAP executesLUFUnaffectedLVUnaffectedUFUnaffectedXUnaffectedZUnaffectedVUnaffectedCUnaffected					
Mode Bit	OVM operation is not affected by OVM bit value.					
Cycles	5					
Example	None					

Syntax	TSTB	src, dst			
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)				
Opcode					
	31	24	23	1615	870
	000	11010() G	dst	SIC
Word Fields					
		_			
		_	G	src addressing modes	
			00	register (any register in CPU primary-register file	9)
			01	direct	
			10	indirect	
		_	11	immediate	
Operation	dst AN	ID <i>src</i>			
Description	The bi is not I and <i>sr</i>	twise-logical / oaded in any r c operands ar	AND of the register. T re assume	e <i>dst</i> and <i>src</i> operands his allows for nondestru ed to be unsigned integ	is formed, but the result ctive compares. The <i>dst</i> ers.
Status Bits	These	condition flag	js are mo	dified for all destination	registers.
	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the o 1 if a zero ou 0 Unaffected	utput tput is gel	nerated, 0 otherwise	
Mode Bit	OVM o	operation is no	ot affected	by OVM bit value.	
Cycles	1				

Example	TSTB *-AR4(1),R5					
	Before Instruc	After Instructi	After Instruction			
	AR4 80 99C5h		AR4 80 99C5h			
	R5 898h	2200	R5 898h	2200		
	Data at 80 99C4h 767h	1895	Data at 80 99C4h	1895		
	LUF 0		LUF 0			
	LV 0		LV 0			
	UF 0		UF 0			
	N 0		N 0			
	Z 0		Z 1			
	V 0		V 0			
	C 0		C 0			

TSTB3 src2, src1

Operands

src1, src2: type 1 or type 2 three-operand addressing modes

Opcode

Type 1

_312	4 23	161	5 8	7 0
0 0 1 0 0 1 1 1	1 T	0 0 0 0 0	src1	src2

Type 2

31 24 23		1615	87	0
0 0 1 1 0 1 1 1 1	T 0 0	0 0 0 src1	5	src2

Word Fields

Type 1

Т	src1 addressing modes	<i>src2</i> addressing modes
00	register mode (any CPU register)	register mode (any CPU register)
01	indirect mode (disp = 0, 1, IR0, IR1)	register mode (any CPU register)
10	register mode (any CPU register)	indirect mode (disp = 0, 1, IR0, IR1)
11	indirect mode (disp = 0, 1, IR0, IR1)	indirect mode (disp = 0, 1, IR0, IR1)

Type 2

Т	src1 addressing modes	src2 addressing modes
00	register mode (any CPU register)	8-bit signed immediate
01	register mode (any CPU register)	indirect mode *+ARn(5-bit unsigned displacement)
10	indirect mode *+ARn(5-bit unsigned displacement)	8-bit signed immediate
11	indirect mode *+ARn1(5-bit unsigned displacement)	indirect mode *+ARn2(5-bit unsigned displacement)

Operation src1 AND src2

Description The bitwise-logical AND between the *src1* and *src2* operands is performed but is not loaded into any register. This allows for nondestructive compares. The *src1* and *src2* operands are assumed to be unsigned integers. The *src2* immediate-addressing mode is sign-extended.

Although this instruction has only two operands, it is designated as a three-operand instruction because operands are specified in the three-operand format.

14-270

Status Bits	LUF LV UF N Z V C	Unaffected Unaffected 0 MSB of the output 1 if a zero output is generated, 0 otherwise 0 Unaffected
Mode Bit	OVM	operation is not affected by OVM bit value.
Cycles	1	
Example	None	

Syntax	XOR sr	c, dst					
Operands	<i>src:</i> general-addressing modes (G) <i>dst</i> : register (any register in CPU primary-register file)						
Opcode							
	31	24 2	3	16	15	8 7	0
	000	1 1 0 1 0 1	G	dst		src	
Word Fields							
		_				_	
		_	G	src a	ddressing modes	_	
			00	regist CPU	er (any register in primary-register file)		
			01	direct			
			10	indire	ct		
		_	11	imme	diate	_	
Operation	dst XOF	R src \rightarrow dst					
Description	The bity register	wise-exclusive . The <i>dst</i> and	e OR of t <i>src</i> oper	he <i>src</i> ands a	and <i>dst</i> operands in are assumed to be u	s loaded into the a nsigned integers.	dst
Status Bits	If ST (S flags ar register	ETCOND) = e modified. If s.	0 and the ST (SET)	e destir COND	nation register is R0) = 1, they are modif	– R11, the conditivited for all destination	on on
	LUF (LV (UF (N N Z 1	Jnaffected Jnaffected) MSB of the ou I if a zero out	itput put is ger	nerated	d, 0 otherwise		
	ν (C () Jnaffected					
Mode Bit	OVM op	peration is no	t affectec	ا by O	VM bit value.		
Cycles	1						

XOR R1,R2

	Before Instruction
R1	0F FA32h
R2	0F F5C1h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

	After Instruction
R1	0F F412h
R2	00 0FF3h
LUF	0
LV	0
UF	0
Ν	0
Z	0
V	0
С	0

Syntax	XOF	R3 src2, src1, ds	t			
Operands	<i>src1, src2:</i> type 1 or type 2 three-operand addressing modes <i>dst:</i> register mode (any register in CPU primary-register file)					
Opcode						
	Туре	e 1				
	31	24 2	23	161	5 8	7 0
	0 0	1 0 1 0 0 0 0) Т	dst	src1	src2
	Туре	e 2				
	31	24 2	23	161	5 8	7 0
	0 0	1 1 1 0 0 0 0) Т	dst	src1	src2
Word Fields						
Type 1						
Турет		cro1 addrossing	modo		cro2 addrossing	modos
	-00	rogistor mode (a)		rogistor)	register mode (ap)	
	00					
	01	$\frac{1}{1000} = 0, 1, 100, 100$		register mode (any	y CPU register)	
	10	register mode (any CPU register)			indirect mode (dis	p = 0, 1, IR0, IR1)
	11	indirect mode (disp = 0, 1, IR0, IR1)			indirect mode (dis	p = 0, 1, IR0, IR1)
Туре 2						
	Т	src1 addressing	g mode	S	src2 addressing	modes
	00	register mode (any CPU register)		8-bit signed immediate		
	01	register mode (any CPU register)		indirect mode *+ARn(5-bit unsigned displacement)		
	10	indirect mode *+ARn(5-bit unsigned displacement)		8-bit signed immediate		
	11	indirect mode *+ARn1(5-bit unsigned displacement)		indirect mode *+ARn2(5-bit unsigned displacement)		
Operation	src1	XOR src2 \rightarrow ds	st			

Description	The bitwise-exclusive OR between the <i>src1</i> and <i>src2</i> operands is loaded into the <i>dst</i> register. The <i>src1</i> , <i>src2</i> , and <i>dst</i> operands are assumed to be unsigned integers. The <i>src2</i> immediate-addressing mode is sign-extended.				
Status Bits	If ST (SETCOND) = 0 and the destination register is $R0 - R11$, the condition flags are modified. If ST (SETCOND) = 1, they are modified for all destination registers.				
	LUFUnaffectedLVUnaffectedUF0NMSB of the outputZ1 if a zero output is generated, 0 otherwiseV0CUnaffected				
Mode Bit	OVM operation is not affected by OVM bit value.				
Cycles	1				
Example	None				

Syntax	XOR3 STI	src2, src1, d src3, dst2	st1			
Operands	src1:register (R0 - R7)src2:indirect (disp = 0, 1, IR0, IR1) $dst1:$ register (R0 - R7)src3:register (R0 - R7) $dst2:$ indirect (disp = 0, 1, IR0, IR1)					
Opcode						
	31 1 1 1 0	24 23 1 1 1 <i>dst</i>	src1	16 <i>src3</i>	15 8 dst2	7 0 src2
Word Fields	None					
Operation	src1 > src3 -	$OR \ src2 \rightarrow dst$ $\rightarrow dst2$	1			
Description	A bitwise-exclusive XOR and an integer store are performed in parallel. All reg- isters are read at the beginning and loaded at the end of the execute cycle. This means that if one of the parallel operations (STI) reads from a register and the operation being performed in parallel (XOR3) writes to the same register, then STI accepts as input the contents of the register before it is modified by the XOR3.					
	If <i>src2</i> and	d <i>dst2</i> point to th	e same loo	cation, <i>sr</i>	<i>c2</i> is read befo	re the write to <i>dst2</i>
Status Bits	 LUF Unaffected LV Unaffected UF 0 N MSB of the output Z 1 if a zero output is generated, 0 otherwise V 0 C Unaffected 					
Mode Bit	OVM operation is not affected by OVM bit value.					
Cycles	1					

2	XOR3	*ARI+	+,R3,R3
\$	STI	R6,*-	AR2(IRO)
		Before I	nstruction
AR1		80 987Eh	
R3		85h	
R6		0DCh	
AR2		80 98B4h	
IR0		8h	
Data	<u>at 80 9</u>	87Eh	
		85h	
Data	<u>at 80 9</u>	8ACh	
		0h	
LUF		0	
LV		0	

UF

Ν

Ζ

٧

С

0

0

0

0

0

Example

	After Ins	truction
AR1	80 987Fh	
R3	0h	
R6	0DCh	
AR2	80 98B4h	
IR0	8h	
Data	at 987Eh	
	85h	
Data	at 80 98ACh	
	0DCh	
LUF	0	
LV	0	
UF	0	
Ν	0	
Z	0	
V	0	
С	0	

220

220

220

Appendix A

Glossary

A

- **A0–A30:** External address pins for data/program memory or I/O devices. These pins are on the global bus. *See also LA0–LA30*.
- address: The location of program code or data stored in memory.
- **addressing mode:** The method by which an instruction interprets its operands to acquire the data it needs.
- ALU: See Arithmetic logic unit.
- **analog-to-digital (A/D) converter:** A successive-approximation converter with internal sample-and-hold circuitry used to translate an analog signal to a digital signal.
- **ARAU:** See auxiliary-register arithmetic unit.
- arithmetic logic unit (ALU): The part of the CPU that performs arithmetic and logic operations.
- auxiliary registers (ARn): A set of registers used primarily in address generation.
- **auxiliary-register arithmetic unit (ARAU):** Auxiliary-register arithmetic unit. A 32-bit arithmetic logic unit (ALU) used to calculate indirect addresses using the auxiliary registers as inputs and outputs.

B

- **bit-reversed addressing:** Addressing in which several bits of an address are reversed in order to speed processing of algorithms, such as Fourier transforms.
- BK: See block-size register.
- **bootloader:** An on-chip code that transfers code from an external memory or from a communication port to RAM at power-up.
- **carry bit:** A bit in status register ST used by the ALU for extended arithmetic operations and accumulator shifts and rotates. The carry bit can be tested by conditional instructions.
- **circular addressing:** An addressing mode in which an auxiliary register is used to cycle through a range of addresses to create a circular buffer in memory.
- **context save/restore**: A save/restore of system status (status registers, accumulator, product register, temporary register, hardware stack, and auxiliary registers, etc.) when the device enters/exits a subroutine such as an interrupt service routine.
- **CPU**: *Central processing unit*. The unit that coordinates the functions of a processor.
- **CPU cycle:** The time it takes the CPU to go through one logic phase (during which internal values are changed) and one latch phase (during which the values are held constant).

cycle: See CPU cycle.

D0–D31: External data-bus pins that transfer data between the processor and external data/program memory or I/O devices. *See also LD0–LD31*.

- **data-address generation logic:** Logic circuitry that generates the addresses for data-memory reads and writes. This circuitry can generate one address per machine cycle. See also *program address generation logic*.
- **data-page pointer:** A 32-bit register used as the 16 MSBs in addresses generated using direct addressing.
- **decode phase:** The phase of the pipeline in which the instruction is decoded (identified).
- DIE: See DMA interrupt enable register.
- **DMA coprocessor:** A peripheral that transfers the contents of memory locations independently of the processor (except for initialization).

DMA controller: See DMA coprocessor.

- **DMA interrupt enable register (DIE):** A register (in the CPU register file) that controls which interrupts the DMA coprocessor responds to.
- **DP:** See data-page pointer.
- **dual-access RAM**: Memory that can be accessed twice in a single clock cycle. For example, your code can read from and write to a dual-access RAM in one clock cycle.

external interrupt: A hardware interrupt triggered by a pin.

- **extended-precision floating-point format:** A 40-bit representation of a floating-point number with a 32-bit mantissa and an 8-bit exponent.
- extended-precision register: A 40-bit register used primarily for extendedprecision floating-point calculations. Floating-point operations use bits 39–0 of an extended-precision register. Integer operations, however, use only bits 31–0.
- **FIFO buffer:** *First-in, first-out buffer.* A portion of memory in which data is stored and then retrieved in the same order in which it was stored. Thus, the first word stored in this buffer is retrieved first. The 'C4x's communication ports each have two FIFOs: one for transmit operations and one for receive operations.
- **hardware interrupt:** An interrupt triggered through physical connections with on-chip peripherals or external devices.
- **hit:** A condition in which, when the processor fetches an instruction, the instruction is available in the cache.
- **IACK:** Interrupt acknowledge signal. An output signal that indicates that an interrupt has been received and that the program counter is fetching the interrupt vector that will force the processor into an interrupt service routine.

Glossary A-3

- **IIE:** See internal interrupt enable register.
- **IIF:** See IIOF flag register.
- **IIOF flag register (IIF):** Controls the function (general-purpose I/O or interrupt) of the four external pins (IIOF0 to IIOF3). It also contains timer/DMA interrupt flags.
- index registers: Two registers (IR0 and IR1) that are used by the ARAU for indexing an address.
- **internal interrupt:** A hardware interrupt caused by an on-chip peripheral.
- **internal interrupt enable register:** A register (in the CPU register file) that determines whether or not the CPU will respond to interrupts from the communication ports, the timers, and the DMA coprocessor.
- **interrupt:** A signal sent to the CPU that (when not masked) forces the CPU into a subroutine called an interrupt service routine. This signal can be triggered by an external device, an on-chip peripheral, or an instruction (TRAP, for example).
- **interrupt acknowledge (IACK):** A signal that indicates that an interrupt has been received, and that the program counter is fetching the interrupt vector location.
- **interrupt vector table (IVT):** An ordered list of addresses which each correspond to an interrupt; when an interrupt occurs and is enabled, the processor executes a branch to the address stored in the corresponding location in the interrupt vector table.
- **interrupt vector table pointer (IVTP):** A register (in the CPU expansion register file) that contains the address of the beginning of the interrupt vector table.
- **ISR:** Interrupt service routine. A module of code that is executed in response to a hardware or software interrupt.
- **IVTP:** See interrupt vector table pointer.
- **LA0–LA30:** External address pin*s* for data/program memory or I/O devices. These pins are on the local bus. *See also A0–A30*.
- **LD0–LD31:** External data bus pins that transfer data between the processor and external data/program memory or I/O devices. *See also D0–D31*.

LSB: Least significant bit. The lowest order bit in a word.

machine cycle: See CPU cycle.

- **mantissa:** A component of a floating-point number consisting of a fraction and a sign bit. The mantissa represents a normalized fraction whose binary point is shifted by the exponent.
- **maskable interrupt**: A hardware interrupt that can be enabled or disabled through software.
- **memory-mapped register:** One of the on-chip registers mapped to addresses in memory. Some memory-mapped registers are mapped to data memory, and some are mapped to input/output memory.
- **MFLOPS:** *Millions of floating point operations per second.* A measure of floating-point processor speed that counts of the number of floating-point operations made per second.
- **microcomputer mode:** A mode in which the on-chip ROM (bootloader) is enabled. This mode is selected via the MP/MC pin. See also MP/MC pin; *microprocessor mode*.
- **microprocessor mode:** A mode in which the on-chip ROM is disabled. This mode is selected via the MP/MC pin. See also MP/MC pin; microcomputer mode.
- **MIPS**: Million instructions-per-second.
- **miss:** A condition in which, when the processor fetches an instruction, it is not available in the cache.
- **MSB**: *Most significant bit.* The highest order bit in a word.
- **multiplier:** A device that generates the product of two numbers.

Ν

- **NMI:** See Nonmaskable interrupt.
- **nonmaskable interrupt (NMI):** A hardware interrupt that uses the same logic as the maskable interrupts but cannot be masked.

Glossary A-5

0

overflow flag (OV) bit: A status bit that indicates whether or not an arithmetic operation has exceeded the capacity of the corresponding register.

PC: See program counter.

peripheral bus: A bus that is used by the CPU to communicate the DMA coprocessor, communication ports, and timers.

pipeline: A method of executing instructions in an assembly-line fashion.

- **program counter:** A register that contains the address of the next instruction to be fetched.
- - RC: See repeat counter register.
 - **read/write (R/W) pin:** This memory-control signal indicates the direction of transfer when communicating to an external device.
 - register file: A bank of registers.
 - **repeat counter register:** A register (in the CPU register file) that specifies the number of times minus one that a block of code is to be repeated when a block repeat is performed.
 - **repeat mode:** A zero-overhead method for repeating the execution of a block of code.
 - reset: A means to bring the central processing unit (CPU) to a known state by setting the registers and control bits to predetermined values and signaling execution to fetch the reset vector.
 - reset pin: This pin causes the device to reset.
 - **ROMEN:** *ROM enable.* An external pin that determines whether or not the the on-chip ROM is enabled.
 - $\mathbf{R}/\overline{\mathbf{W}}$: See read/write pin.

short floating-point format: A 16-bit representation of a floating point number with a 12-bit mantissa and a 4-bit exponent.

short integer format: A twos-complement,16-bit format for integer data.

short unsigned-integer format: A 16-bit unsigned format for integer data.

- **sign-extend:** Fill the high order bits of a number with the sign bit.
- **single-precision floating-point format:** A 32-bit representation of a floating-point number with a 24-bit mantissa and an 8-bit exponent.
- single-precision integer format: A twos-complement 32-bit format for integer data.
- **single-precision unsigned-integer format:** A 32-bit unsigned format for integer data.
- **software interrupt:** An interrupt caused by the execution of a TRAP instruction.
- **split mode:** A mode of operation of the DMA coprocessor. This mode allows one DMA channel to service both the receive and transmit portions of a communication port.
- ST: See status register.
- **stack:** A block of memory reserved for storing and retrieving data on a first-in last-out basis. It is usually used for storing return addresses and for preserving register values.
- **status register:** A register in the CPU register file that contains global information related to the CPU.

Timer: A programmable peripheral that can generate pulses or time events.

- **Timer-Period Register:** *Timer-period register.* A 32-bit memory-mapped register that specifies the period for the on-chip timer.
- **trap vector table (TVT):** An ordered list of addresses which each correspond to an interrupt; when a trap is executed, the processor executes a branch to the address stored in the corresponding location in the trap vector table.
- trap vector table pointer (TVTP): A register in the CPU expansion register file that contains the address of the beginning of the trap vector table.

Glossary A-7

	TVTP: See trap vector table pointer.
U	unified mode: A mode of operation of the DMA coprocessor. The mode is used mainly for memory-to-memory transfers. This is the default mode of operation for a DMA channel. See also <i>split mode</i> .
W	
	wait state : A period of time that the CPU must wait for external program, data, or I/O memory to respond when reading from or writing to that external memory. The CPU waits one extra cycle for every wait state.
	wait-state generator : A program that can be modified to generate a limited number of wait states for a given off-chip memory space (lower program, upper program, data, or I/O).
Ζ	

zero fill: Fill the low or high order bits with zeros when loading a number into a larger field.

Nu

16-bit wide configured memory table 10-1432-bit wide configured memory table 10-15

Α

A/D converter definition A-1 A0-A30 definition A-1 abbreviations 14-16 ABS||STI instruction 14-31 ABSF instruction 14-26 ABSF||STF instruction 14-27 ABSI instruction 14-29 ADDC instruction 14-33 ADDC3 instruction 14-35 ADDF instruction 14-37 ADDF3 instruction 14-39 ADDF3||STF instruction 14-41 ADDI instruction 14-43 ADDI3 instruction 14-45 ADDI3||STI instruction 14-47 addition floating-point 5-23 address definition A-1 address buses external 2-20 address partitioning figure 4-10 address pins external A-4

address range LSTRB0 9-11 STRB0 9-10 address space caution 2-13 addressing modes bit-reversed addressing 6-32 circular 6-27 conditional branch 2-18 definition A-1 encoding 6-21 conditional branch 6-25 general 6-21 parallel 6-24 three-operand 6-22 general 2-18 groups 6-21 parallel 2-18 three operand 2-18, 6-22 addressing types 6-2 direct addressing 6-5 immediate 6-18 indirect addressing 6-6 to 6-21 PC relative 6-19 register 6-3 AE bit 9-7 aliasing 2-17 ALU. See arithmetic logic unit; arithmetic logic unit (ALU) analysis bit 3-7 analysis module registers 4-6 AND instruction 14-49 AND||STI instruction 14-53 AND3 instruction 14-51 ANDN instruction 14-55 ANDN3 instruction 14-57

application(s) automotive viii, xiv consumer viii. xiv control viii, xii development support viii, xv general-purpose viii graphics/imagery viii, xi medical viii, xiv military viii, xiii multimedia viii, xiii speech/voice viii, xi telecommunications viii, xiii ARAUs. See auxiliary register arithmetic units (ARAUs) architectural overview introduction 2-1 architecture peripheral bus 2-22 arithmetic logic unit (ALU) 2-4 definition A-1 ASH instruction 14-59 ASH3 instruction 14-61 ASH3||STI instruction 14-63 assembly language instructions 14-2 to 14-11 condition codes flags 14-12 example instruction 14-23 illegal instructions 14-11 interlocked operation 14-8 load and store 14-2 parallel operation 14-9 program control 14-7 register symbols 14-21 to 14-22 symbols 14-17 to 14-22 syntax options 14-18 to 14-22 three-operand 14-6 two-operand 14-4 autoinitialization 11-34 consecutive 11-40 situations 11-34 synchronization 11-37 automotive applications viii, xiv auxiliary link-pointer register 11-7 auxiliary register indirect 6-9 auxiliary register arithmetic units (ARAUs) 2-6 auxiliary registers (AR0-7) 2-6, 3-4

auxiliary registers (ARn) definition A-1 auxiliary transfer-counter register 11-7 auxiliary-register arithmetic unit (ARAU) definition A-1

Β

Bcond instruction 14-65 BcondAF instruction 7-11, 8-7, 14-67 example 8-7 BcondAT instruction 7-11, 8-7, 14-69 example 8-7 BcondD instruction 14-71 bit-reversed addressing 6-32 definition A-1 example 6-32 index steps 6-33 block diagram 'C4x 2-2 communication ports 12-4 peripheral modules 2-22 timers 13-3 block repeat nesting 7-8 registers (RC, RE, RS) 3-16, 7-2 block size (BK) register 3-5 block transfer completion 11-6 block transfer sequence 11-5 bootloader definition A-2 description 10-2 from communication port 10-3 from memory 10-3 introduction 10-1 mode selection 10-3 operation 10-5 setting the IIOF pins 10-19 source code 10-20 to 10-25 source structure 10-8 bootloader mode selection table 10-3 bootloading from a comm port 10-16 from memory 10-10 sequence 10-5 BR instruction 14-73 branch conflicts 8-4

branch execution delayed 7-10 branches 7-9, 7-12 BRD instruction 14-74 bus operation external 2-20, 9-1 to 9-50 internal 2-19 busy-waiting example 9-42 byte-wide configured memory table 10-11 to 10-14

С

C flag 3-5 'C40 memory map figure 2-14, 4-3 'C40/'C44 features table 1-4 'C44 memory aliasing figure 2-17 'C44 memory map 4-4 figure 2-15 'C4x multiprocessor system booting example 10-17 'C4x to IEEE conversion example 5-18 'C4x-specific instructions 14-3 to 14-8 cache 4-1 cache clear (CC) bit 4-12 cache enable (CE) bit 4-12 cache freeze (CF) bit 4-12 cache memory 2-11, 4-13 architecture 2-11, 4-10 control bits 4-12 enabling 4-13 hit 4-14 instruction cache 4-10 LRU algorithm 4-14 miss 4-14 segment miss 4-14 subsegment miss 4-14 CALL instruction 7-12, 14-75 CALL response timing figure 7-14 CALLcond instruction 7-12, 14-76 calls 7-12

carry bit definition A-2 CC bit 3-6, 4-12 CE and CF bits combined effect table 4-13 table 3-7 CE bit 3-6, 4-12 CE0 bit 9-7 CE1 bit 9-7 central processing unit. See CPU CF bit 3-6, 4-12 channel control register. See DMA channel control register channel priority scheme split mode 11-25 circular addressing definition A-2 example 6-30 FIR filters 6-31 register relationships figure 6-28 circular addressing mode 6-27 circular buffer implementation 6-29 CLKSRC = 0 and FUNC = 0 13-14 CLKSRC = 0 and FUNC = 1 13-14 CLKSRC = 1 and FUNC = 0 13-13 CMPF instruction 14-78 CMPF3 instruction 14-80 CMPI instruction 14-82 CMPI3 instruction 14-84 communication port load mode flow chart 10-7 communication port memory map figure 12-7 communication port reset example 12-10 communication port software register 12-3 communication ports arbitration unit 12-3, 12-11 block diagram 12-4 control register 12-3 coordination with CPU/DMA 12-17 CSTRB width restrictions 12-25 features 2-23, 12-2 H1/H3 synchronization 12-26 input FIFO halt 12-15

communication ports (continued) input port post-reset state 12-31 input port register 12-9 interconnection 12-5 introduction 12-1 memory map 4-8, 12-7 figure 4-8 output FIFO halt 12-15 output port post-reset state 12-30 output port register 12-9 reset 12-29 tips 12-32 token transfer 12-19 communication-port control register (CPCR) 12-8 field descriptions 12-8 figure 12-8 ICH 12-8 INPUT LEVEL 12-9 OCH 12-8 OUTPUT LEVEL 12-9 PORT DIR 12-8 communication-port software reset address table 12-10 reset register 12-10 condition codes flags 14-14 conditional-branch addressing modes 2-18, 6-25 encoding 6-26 consecutive autoinitializations 11-40 consumer applications viii, xiv context save/restore definition A-2 control applications viii, xii control bits repeat mode 7-3 control registers 7-35, 11-7, 13-5 conversion of format 'C4x floating-point to integer 5-31 extended-precision floating-point to single-precision floating-point 5-12 FRIEEE instruction 14-98 IEEE single precision std. 754 5-13 IEEE to 'C4x floating-point 5-14 integer to floating-point 5-33 short floating-point to extended-precision floatingpoint 5-11 short floating-point to single-precision floatingpoint 5-11

single-precision 'C4x floating-point 5-13 single-precision floating-point to extended-precision floating-point 5-12 TOIEEE instruction 14-265 converting IEEE format table 5-14 converting twos complement table 5-17 counter register 13-5 CPU 2-4 arbitration 11-27 block diagram 2-5 buses 2-19 components 2-4 communication ports coordination 12-17 definition A-2 internal interrupt enable register (IIE) 2-9, 3-11 primary register file 2-6 CPU cycle definition A-2 CPU expansion register file definition 3-1 CPU primary register file 3-2 definition 3-1 CPU registers 2-7, 3-8, 7-36 auxiliary (AR0-AR7) 2-6, 3-4 block repeat (RC, RE, RS) 3-16 block size (BK) 2-8, 3-5 data page pointer (DP) 2-8, 3-4, 6-5 DMA interrupt enable (DIE) 2-9, 3-8, 11-44 expansion register file 2-10, 3-17 extended precision (R0-R11) 2-6, 3-3 IIE 3-11 IIOF flag register (IIF) 2-9, 3-13 index (IR1, IR0) 2-8, 3-4 internal interrupt enable (IIE) 2-9, 3-11, 3-12 introduction 3-1 program counter (PC) 2-9, 2-19, 3-16 repeat count (RC) 2-9, 3-16, 7-2 repeat end address (RE) 3-16, 7-2 See also repeat block (RC, RE, RS) repeat start address (RS) 3-16, 7-2 See also repeat block (RC, RE, RS) stack pointer (SP) 2-8, 3-5 status register (ST) 2-9, 3-5, 14-13 table 3-2, 6-3 timer 4-7 CSTRB width restrictions 12-25

D

D0-D31 definition A-2 data buses external 2-20 data formats introduction 5-1 data page pointer (DP) 2-8, 3-4, 6-5 data structure FIR filters 6-31 data transfer modes 11-28 data transfer operation 12-6 data-address generation logic definition A-2 data-page pointer definition A-2 DBcond instruction 14-86 DBcondD instruction 14-88 DBR instruction 8-9 DE bit 9-7 decode phase definition A-2 delayed branches 7-9 conditional 7-9 disabled interrupts 7-9 example 7-10 incorrectly placed 7-10 example 7-7 with annul option 8-7 with annulling 7-11 without annul option 8-6 example 8-6 without annulling 7-10 destination address register 11-7 destination address-index register 11-7 development support applications viii, xv DIE register bit functions DMA split mode 11-45 direct addressing 6-5 example 6-5 figure 6-5 direct memory access. See DMA coprocessor displacement indirect addressing table 6-7

displacements 6-6 to 6-21 DMA channel control register 7-35, 11-7 bit definitions 11-8 data transfer modes 11-28 field descriptions 11-8 modifiable by autoinitialization in split mode 11-40 modifiable by autoinitialization in unified mode 11-39 modifiable by autoinitialization of auxiliary channel 11-40 DMA channel control register bit definitions AUTOINIT STATIC 11-8 AUTOINIT SYNC 11-9 AUX AUTOINIT STATIC 11-9 AUX AUTOINIT SYNC 11-9 AUX START 11-10 AUX STATUS 11-11 AUX TCC 11-10 AUX TCINT FLAG 11-10 AUX TRANSFER MODE 11-8 COM PORT 11-9 DMA PRI 11-8 PRIORITY MODE 11-11 READ BIT REV 11-9 SPLIT MODE 11-9 START 11-10 STATUS 11-11 SYNC MODE 11-8 TCC 11-10 TCINT FLAC 11-10 TRANSFER MODE 11-8 WRITE BIT REV 11-9 DMA channel registers (SPLIT MODE=1, auxiliary transfer counter = 0) figure 11-36 storage in memory (SPLIT MODE=0) figure 11-35 storage in memory (SPLIT MODE=1) figure 11-36 DMA channel running transfer mode 102 figure 11-29, 11-30 transfer mode 112 figure 11-31, 11-33 DMA control register bits effect 11-38 DMA controller 2-19

DMA coprocessor 2-23 address generation fiaure 11-16 address registers 11-15 arbitration 11-27 autoinitialization 11-34 auxiliary channel 11-20 block transfer sequence 11-5 buses 2-19 channel arbitration 11-24 channel configuration figure 11-19 channel register map 4-9, 11-4 channel synchronization 11-43 to 11-46 communication ports coordination 12-17 definition A-2 destination synchronization 11-48 features 11-2 functional description 11-3 index registers 11-15 interrupts 11-42 introduction 11-2 link-pointer register 11-17 memory map **11-4** operational modes 11-3 primary channel 11-20 priorities 11-22 priority wheel 11-24 registers 11-5, 11-7 six channels 2-23 source and destination synchronization 11-49 source synchronization 11-48 split mode 3-10, 11-20 timing 11-51 transfer count register 11-16 transfer modes 11-28 unified mode 3-8, 11-19 DMA coprocessor memory map figure 4-9 DMA destination synchronization figure 11-49 DMA interrupt enable register (DIE) 2-9, 3-8, 11-44 bit functions 3-10 definition A-3 DMA interrupts 7-26 control bits 7-26 CPU interaction 7-28 processing 7-27 DMA memory transfer timing single 11-51

DMA PRI and CPU/DMA arbitration rules 11-27 table 11-12 DMA registers intialization 11-5 DMA source and destination sync unified mode 11-50 DMA source synchronization figure 11-48 DMA start 11-5 DMA timing for different synchronizations split mode figure 11-56 unified mode figure 11-55 DMA transfers timing and number of cycles to global bus figure 11-54 timing and number of cycles to local bus figure 11-53 timing and number of cycles to on-chip figure 11-52 DMAINTx flag 3-15 documentation vii dual-access RAM definition A-3

Ξ

edge-triggered interrupts 7-15 EDMAINTx bit 3-12 EICFULLx bit 3-12 EICRDYx bit 3-12 EIIFOx bit 3-14 EOCEMPTYx bit 3-12 EOCRDYx bit 3-12 ETINT0 bit 3-12 ETINT1 bit 3-12 execute only 8-10, 8-13 parallel store 8-14 single store 8-13 expansion register file 2-10, 3-17 interrupt vector table (IVT) 7-16 exponent field definition 5-8 extended precision registers 3-3 extended-precision floating-point format definition A-3

extended-precision register definition A-3 floating-point format 3-4 integer format 3-4 external bus control registers. *See* memory interface control registers interface signals 9-3 external bus operation 9-1 to 9-50 overview 9-2 external buses (global, local), wait states 9-14 external interrupts 2-21, **7-21** definition A-3 external memory interface registers 7-35

F

features comparison 1-4 FIFO buffer definition A-3 **FIFOS** halting 12-14 FIR filters circular addressing 6-31 data structure 6-31 FIX instruction 5-31, 14-90 FIX||STI instruction 14-92 fixed priority 11-22 FLAGx bit 3-14 FLOAT instruction 5-33, 14-94 FLOAT||STF instruction 14-96 floating point addition 5-23 conversion to integer 5-31 extended-precision format 5-7 format conversion 5-11 formats 5-4 normalization 5-23, 5-27 reciprocal 5-34 register format 3-4 rounding value 5-29 single-precision format 5-6 floating point (continued) subtraction 5-23 underflow 5-24

floating-point determining decimal equivalent 5-8 extended-precision format 5-7 general format 5-4 multiplication 5-19 short format 5-5 figure 5-5 single-precision format figure 5-6 floating-point addition 32-bit shift 5-26 example 5-25 floating-point addition/subtraction example 5-26 floating-point formats IEEE Std. 754 5-13 supported types 5-4 floating-point multiplication chart 5-20 floating-point multiply mantissa = 1.0 5-22 mantissa = 1.5 5-21 mantissa = 2.0 5-21 positive and negative numbers 5-22 floating-point operation introduction 5-1 floating-point rounding flowchart 5-30 floating-point subtraction example 5-25 floating-point to integer conversion flowchart 5-32 floating-point values fractional 5-10 negative 5-10 positive 5-9 floating-point/integer multiplier 2-4 format conversion 'C4x to IEEE 5-17 conversions IEEE std. 754 5-13 formats conversion floating-point 5-11 See also conversion of formats formats (continued) signed integer 5-2 unsigned integer 5-3

FRIEEE instruction 14-98 FRIEEE||STF instruction 14-99 FUNCx bit 3-14

G

general addressing modes 2-18, 6-21 encoding 6-22 general-purpose applications viii GIE bit 3-7 global and local memory interface control signals 9-3 global memory 9-39, 9-41, 9-43 interface 2-20, 9-2 *table 9-4* global memory port status STRB0 and STRB1 accesses 9-5 graphics/imagery applications viii, xi

Η

halting of FIFOs 12-14 hardware interrupt definition A-3 hit cache 4-14 definition A-3 hold everything 8-10, 8-15 busy external port 8-15 conditional calls and traps 8-16 multicycle data reads 8-16

IACK definition A-3 IACK instruction 9-49, 14-100 IACK pin 9-49 timing 9-49 ICFULL interrupt description 12-17 enabling 3-12 ICRDY flag interrupt use 11-46 ICRDY interrupt description 12-17

enabling 3-12 interrupt use 3-9, 3-10, 11-44, 11-45 IDLE instruction 14-102, 14-103 IEEE std. 754 (conversions) 5-13 IEEE to 'C4x conversion example 5-16 IIE register 3-11 IIF register 7-17 IIF register modification 3-13 figure 7-18 IIOF flag register (IIF) 2-9, 3-13 definition A-4 **IIOF** pins boot loader use 10-5 modification 10-19 immediate addressing 6-18 example 6-18 index registers (IR0, IR1) 2-8, 3-4, 11-15 definition A-4 indirect addressing 6-6 displacement 6-7 flexibility 6-6 index register IR1 6-8 index register IRO 6-7 operand coding figure 6-6 postdisplacement add and circular modify 6-12 postdisplacement add and modify 6-11 postdisplacement subtract and circular modify 6-13 postdisplacement subtract and modify 6-12 postindex add and bit-reversed modify 6-17 postindex add and circular modify 6-16 postindex add and modify 6-15 postindex subtract and circular modify 6-17 postindex subtract and modify 6-16 predisplacement add 6-9 predisplacement add and modify 6-10 predisplacement subtract 6-10 predisplacement subtract and modify 6-11 preindex add 6-13 preindex add and modify 6-14 preindex subtract 6-14 preindex subtract and modify 6-15 special cases 6-8 individual instruction descriptions 14-20 input and output FIFO halting summary 12-14

input FIFO channel 12-3

instruction register (IR) 2-19

instruction set summary 14-2 to 14-11 functional groups 14-2 instructions See also assembly language interlocked 9-44 integer short format 5-2 short unsigned format 5-3 signed formats 5-2 single-precision unsigned format 5-3 single-precision format 5-2 unsigned formats 5-3 integer formats short integer 5-2 signed 5-2 interlocked instructions 2-20, 9-39, 9-44 interlocked operations 9-39 interlocked operations instructions table 14-8 internal buses 2-4, 2-19 internal interrupt definition A-4 internal interrupt enable register (IIE) 2-9, 3-11 definition A-4 internal interrupts 7-18 interrupt definition A-4 interrupt acknowledge (IACK) definition A-4 interrupt acknowledge (IACK) instruction 7-20 interrupt flag register (IIF) figure 3-14 interrupt latency table 7-21 interrupt service routine (ISR) definition A-4 interrupt vector table (IVT) 7-15 boot loader use 10-8 definition A-4 interrupt vector table pointer (IVTP) definition A-4

instruction cache 4-10

architecture 4-10

fiaure 4-11 reset 4-12

interrupts 2-21 control bits 7-17 DMA 7-28, 11-42 DMA interaction 7-28 edge triggered 11-42 edge-triggered 11-42 external 2-21, 7-21 initialization 7-24 initiation condition 7-15 latency 7-20 level triggered 11-42 level-triggered 11-42 NMI 7-22 overlapping the IVT and TVT 7-25 prioritization 7-15 processing 7-18, 7-19, 7-27, 7-28 timer 13-12 vector table 7-16 vectors 7-28, 13-11 ISR. See interrupt service routine (ISR) IVTP. See interrupt vector table (IVT) IVTP register 2-10, 3-17



jumps 7-12



LA0-LA30 definition A-4 LAJ instruction 7-13. 14-105 LAJcond instruction 7-13, 14-106 LATcond instruction 7-13, 7-25, 14-107 LBb instruction 14-108 LBUb instruction 14-110 LD0-LD31 definition A-4 LDA instruction 14-111 LDE instruction 14-112 LDEP instruction 14-114 LDF instruction 14-115 LDF||LDF instruction 14-121 LDF||STF instruction 14-123 LDFcond instruction 14-117 LDFI instruction 9-39, 9-45, 14-119 LDHI instruction 14-125

LDI instruction 14-126 LDI||LDI instruction 14-132 LDI||STI instruction 14-134 LDIcond instruction 14-128 LDII instruction 9-39, 9-45, 14-130 LDM instruction 14-136 LDP instruction 14-138 LDPE instruction 14-139 LDPK instruction 14-140 level-triggered interrupts 7-15 LHUw instruction 14-143 LHw instruction 14-141 link pointer incrementing 11-36 reference to 11-41 self referential 11-41 link pointer registers 11-7 figure 11-18 literature vii LLOCK signal 9-44 load and store instructions table 14-3 loading sequence bootloader 10-10 local memory interface 2-20, 9-2 LOCK signal 9-44 low IIOF signal circuit diagram 10-19 LRU algorithm 4-14 LRU stack 4-12 LSB definition A-5 LSH instruction 14-145 LSH3 instruction 14-147 LSH3||STI instruction 14-149 LUF flag 3-5 LV flag 3-5 LWLct instruction 14-152 LWRct instruction 14-154

Μ

machine values 14-21

Index-10

mantissa definition 5-8, A-5 mapping addresses to strobes 9-12 maskable interrupt definition A-5 MBct instruction 14-156 medical applications viii, xiv memory 2-11, 8-10 See also memory interface accesses pipeline 8-19 timing 8-19 addressing modes 2-18 aliasing 2-17 block diagram 2-12 cache 4-10, 4-13 communication ports memory map 12-7 control registers. See memory interface global 9-39, 9-41, 9-43 introduction 4-1 maps 2-13, 4-2 maximizing pipeline performance 8-17 memory maps communication ports 4-8 DMA 11-4 timer registers 4-7, 13-5 organization 2-11, 4-2 parallel multiplies and adds 8-23 parallel stores 8-21 pipeline conflicts 8-10 ranges 9-10 registers. See memory interface control registers ROMEN pin 4-2 sharing 9-42 signal-group control 9-38 space 2-11 three-operand accesses 8-20 timing 8-19, 9-16 two-operand accesses 8-20 memory accesses data access 8-17 data loads and stores 8-20 external program fetches 8-19 internal clock 8-19 internal program fetches 8-19 program fetch 8-17 two data accesses 8-18 memory cache rules for efficient usage 4-13

memory conflicts 8-4 memory interface address ranges 9-11 control registers 9-6 control signals 9-3 page size 9-9 PAGESIZE field. See memory interface control registers memory interface (local, global) features 9-2 ready generation 9-14, 9-16 timing 9-16 wait states 9-14 memory interface control registers 4-6 address ranges 9-10 bit contents 9-7 fields figure 9-7 figure 4-6 reset effect 9-6 STRBx SWW field 9-15 timing 9-16 wait states 9-14 memory load flow chart 10-6 memory map 4-2 analysis module registers 4-6 'C44 2-15 communication ports 4-8, 12-7 DMA coprocessor figure 4-9 DMA 4-9, 11-4 global memory bus 9-12 peripheral 2-16 timer registers 4-7, 13-5 memory-mapped register definition A-5 **MFLOPS** definition A-5 MHct instruction 14-158 microcomputer mode definition A-5 microprocessor mode definition A-5 military applications viii, xiii MIPS definition A-5

miss cache 4-14 definition A-5 mode selection bootloader 10-3 mode selection flow figure 10-4 module reset 12-29 MPYF instruction 14-160 MPYF3 instruction 14-162 MPYF3||ADDF3 instruction 14-163 MPYF3||STF instruction 14-165 MPYF3||SUBF3 instruction 14-167 MPYI instruction 14-169 MPYI3 instruction 14-171 MPYI3||ADDI3 instruction 14-173 MPYI3||STI instruction 14-175 MPYI3||SUBI3 instruction 14-177 MPYSHI instruction 14-179 MPYSHI3 instruction, 14-180 MPYUHI instruction 14-182 MPYUHI3 instruction 14-183 MSB definition A-5 multimedia applications viii, xiii multiplication floating-point 5-19 multiplier definition A-5 multiply or CPU operation parallel store 8-21

Ν

N flag 3-5 NEGB instruction 14-185 NEGF instruction 14-187 NEGF||STF instruction 14-189 NEGI instruction 14-191 NEGI||STI instruction 14-193 Newton-Raphson algorithm example 5-35 reciprocal square root 5-38 NMI 7-22 NMI bus grant field 3-7 NMI flag 3-15

no DMA synchronization figure 11-47 nonmaskable interrupt (NMI) definition A-5 NOP instruction 14-195 NORM instruction 5-27, **14-196** execution 5-28 flowchart 5-27 normalization floating point value 5-23, 5-27 NOT instruction 14-198 NOT||STI instruction 14-200

0

object values three-operand instructions 6-23 **OCEMPTY** interrupt description 12-17 enabling 3-12 OCRDY flag interrupt use 11-46 OCRDY interrupt description 12-17 enabling 3-12 interrupt use 3-9, 3-10, 11-44, 11-45 operational overview communication ports 12-3 OR instruction 14-202 OR3 instruction 14-204 OR3||STI instruction 14-206 output FIFO channel 12-3 output value formats 14-12 overflow 5-24, 5-31 overflow flag (OV) bit definition A-6 OVM flag 3-6

Ρ

P flag (cache) 4-10 page size 9-9 page size operation 9-13 parallel addressing modes 2-18, 6-24 parallel instructions table 14-9 parallel multiplies and adds figure 8-23 parallel multiply with ADD/SUB encoding 6-24 PAU. See port arbitration unit PAU state definitions 12-11 PCF bit 3-6 PC-relative addressing encoding 6-20 example 6-19 period register 13-5 peripheral memory map 4-5 peripheral modules figure 2-22 peripheral bus 2-22 definition A-6 general architecture 2-22 memory map 4-5 peripheral memory map 2-16 peripherals communication port 2-23 PGIE bit 3-7 pin states table 7-29 to 7-35 pipeline conflicts 8-4 branch 8-4 memory 8-10 register 8-8 resolving (memory) 8-17 decode unit 8-2 definition A-6 execute unit 8-2 fetch unit 8-2 four major units 8-2 introduction 8-1 memory accesses 8-19 read unit 8-2 structure 8-2 pipeline structure figure 8-3 POP instruction 14-208 POPF instruction 14-209 port arbitration unit 12-3, 12-11 previous cache freeze (PCF) bit 4-13 primary register file (CPU) 2-6, 3-2

prioritization 7-15

priority wheel (DMA) 11-24 program buses 2-19 program control instructions table 14-7 program counter (PC) 2-9, 2-19, **3-16** definition A-6 program fetch multicycle program memory fetches 8-12 program fetch incomplete 8-10, 8-12 program wait 8-10 due to multicycle access 8-12 wait until CPU data access completes 8-11 PUSH instruction 14-210 PUSHF instruction 14-211

R

RAM 2-11 RC register 7-4 RCPF instruction 5-34, 5-35, 14-212 RCPF instruction algorithm figure 5-34 read of AR 8-9 read/write (R/W) pin definition A-6 ready generation 9-14 timing 9-16 reciprocal (RCPF instruction) 5-34 reciprocal algorithm 5-35 reciprocal square root (RSQRF instruction) 5-36 register addressing 6-3 register bit functions DMA unified mode figure 3-8 register buses 2-19 register conflicts 8-4 register file definition A-6 registers 2-6, 2-7 auxiliary (AR0-AR7) 2-6, 3-4 block repeat (RC, RE, RS) 3-16 block size (BK) 2-8, 3-5 data page pointer (DP) 2-8, 3-4, 6-5

DMA interrupt enable (DIE) 2-9, 3-8 expansion register file 2-10 extended precision 3-3 extended precision (R0-R11) 2-6 IIOF flag register (IIF) 2-9, 3-13, 7-17, 7-26 index (IR1, IR0) 3-4 input port 12-9 internal interrupt enable (IIE) 2-9, 3-11, 3-12 output port 12-9 pipeline conflicts 8-8 program counter (PC) 2-9, 2-19, 3-16 repeat count (RC) 2-9, 3-16, 7-2 repeat end address (RE) 3-16, 7-2 See also repeat block (RC, RE, RS) repeat mode 7-2 repeat start address (RS) 3-16, 7-2 See also repeat block (RC, RE, RS) stack pointer (SP) 2-8, 3-5 status register (ST) 2-9, 3-5, 14-13 timer counter 13-8 timer global control 13-6 repeat count register (RC) 2-9, 3-16, 7-2 definition A-6 repeat end address register (RE) 7-2 repeat mode 7-2 block (RPTB) 7-2 block delayed (RPTBD) 7-2 control bits 7-3 definition A-6 nesting 7-8 operation 7-3 RC value after completion 7-7 restriction rules 7-6 **RPTB** instruction 7-4 **RPTBD** instruction 7-4 RPTS instruction 7-5 single instruction (RPTS) 7-2 repeat mode flag 3-6 repeat mode registers 7-2 repeat start address register (RS) 7-2 repeat-mode control algorithm example 7-4 reserved bits 3-16 reset 7-29 additional operations 7-35 communication ports 12-29 definition A-6

reset (continued) memory interface control registers 9-6 pin states 7-29 vector location 7-35 vectors 7-28 RESET pin 12-29 reset pin definition A-6 RESETLOC pins 10-10 RETIcond instruction 7-13, 7-24, 14-213 RETIcondD instruction 7-13, 7-24, 14-214 RETScond instruction 7-12, 14-215 returns 7-12 RM bit 7-3 RM flag 3-6 RND instruction 5-29, 14-216 ROL instruction 14-218 ROLC instruction 14-219 ROM 2-11, 2-13 ROMEN 2-13 definition A-6 ROMEN pin 2-11, 4-2 ROR instruction 14-221 RORC instruction 14-222 rotating priority 11-22 rotating priority DMA read and write sequence 11-23 rotating priority mode figure 11-23 rounding of floating-point value 5-29 RPTB instruction 7-2, 7-8, 8-5, 14-223 pipeline conflict 7-7 **RPTB** operation example 7-4 RPTBD instruction 7-2, 7-8, 14-224 **RPTS** execution steps 7-6 RPTS instruction 7-2, 8-5, 14-226 RSQRF instruction 5-36, 14-228 algorithm figure 5-37

S

S bit 7-3

Index-14

semaphores 9-43 service sequence split mode priority 11-26 SET COND bit 3-7 short floating-point format definition A-7 short integer format definition A-7 short unsigned-integer format definition A-7 SIGI instruction 9-39, 9-47, 14-229 signed-integer formats 5-2 sign-extend definition A-7 single-precision floating-point format definition A-7 single-precision integer format definition A-7 single-precision unsigned-integer format definition A-7 software interrupt definition A-7 source address register 11-7 source address-index register 11-7 source and destination synchronization 11-49 source synchronization 11-48 speech/voice applications viii, xi split mode 3-10 definition A-7 split mode (DMA) 11-20, 11-35 split-mode DMA configuration figure 11-21 split-mode synchronization interrupts table 3-11 stack definition A-7 stack pointer (SP) 2-8, 3-5 standard (nondelayed) branches 8-4 standard branch 7-9 example 8-5 START field descriptions table 11-14 state diagram port arbitration unit 12-12 STATUS field descriptions table 11-14

status register (ST) 2-9, 3-5, 14-13 definition A-7 figure 3-5 STF instruction 14-230 STF||STF instruction 14-234 STFI instruction 9-39, 9-46, 14-232 STI instruction 14-235 STI||STI instruction 14-237 STII instruction 9-39, 9-46, 14-236 STIK instruction 14-239 STRB ACTIVE 9-8 STRB SWITCH 9-8 STRB0 PAGESIZE 9-8 STRB0 SWW 9-8 STRB0 WTCNT 9-8 STRB1 PAGESIZE 9-8 STRB1 SWW 9-8 STRB1 WTCNT 9-8 STRBx PAGESIZE fields figure 9-13 strobe settings 9-7 strobes 9-12 timing 9-16 style (manual) iv SUBB instruction 14-240 SUBB3 instruction 14-242 SUBC instruction 14-244 SUBF instruction 14-246 SUBF3 instruction 14-248 SUBF3||STF instruction 14-250 SUBI instruction 14-252 SUBI3 14-254 SUBI3||STI instruction 14-256 SUBRB instruction 14-258 SUBRF instruction 14-260 SUBRI instruction 14-262 subtraction floating-point 5-23, 5-25 SWI instruction 14-264 symbols 14-16 symbols (used in manual) iv sync mode transfer rate 11-55 SYNC MODE and AUTOINIT MODE bits

autoinitialization table 11-38 SYNC MODE bits 11-46 SYNC MODE field descriptions split mode table 11-13 unified mode table 11-13 synchronization 11-37 destination 11-48 DMA channels 11-43 source 11-48 source and destination 11-49 synchronization interrupts DMA channels 3-9 synchronizer delays 12-28 synchronizers 12-26

task counter example 9-42 TCLK 13-4 TCLK as an input figure 13-15 TCLK as an output figure 13-15 technical assistance xvi telecommunications applications viii, xiii three-operand addressing modes 2-18, 6-22 encoding for type 1 6-24 three-operand instruction word 8-20 three-operand instructions table 14-6 timer definition A-7 interrupts considerations 13-12 timer clock setup maximum setup 13-16 timer configuration CLKSRC = 0 and FUNC = 0 13-14 CLKSRC = 0 and FUNC = 1 13-14 CLKSRC = 1 and FUNC = 0 13-13 CLKSRC = 1 and FUNC = 1 13-13 timer control register bit summary C/P 13-7 CLKSRC 13-7 **DATIN 13-6**

timer control register bit summary (continued) DATOUT 13-6 FUNC 13-6 GO 13-6 HLD 13-6 I/O 13-6 INV 13-7 **TSTAT 13-7** timer global control register diagram bit summary 13-6 timer output generation examples 13-10 timer pins 13-4 timer pulse mode clock mode timing 13-9 timer registers 4-7, 7-35 figure 4-7 timer-period register definition A-7 timers 2-24, 13-2, 13-2 to 13-3 boundary conditions 13-8 control registers 13-5 counter register 13-2, 13-8 global control register 13-6 I/O pin 2-24 initialization 13-16 interrupts 13-11 operation 13-11 introduction 13-1 operation nodes 13-10 period register 13-2, 13-7 pulse generation 13-9 selecting CLKSRC 13-13 selecting FUNC 13-13 TCLK general-purpose I/O 13-15 timing DMA channels 11-51 IACK pin 9-49 memory access 9-16 STRB, RDY 9-16 TINT0 flag 3-15 TINT1 flag 3-15 TMS320C40 introduction 1-2 TMS320C44 introduction 1-2

TMS320C4x features comparison 1-4 introduction 1-1 key features 1-3 TMS320C4x devices 1-2 TMS320LC40 introduction 1-2 TOIEEE instruction 14-265 TOIEEE||STF instruction 14-266 token 12-3 token transfer figure 12-20 operation 12-5, 12-19 token transfer sequence table 12-21 transfer counter registers figure 11-17 TRANSFER MODE = 002 running 11-28 TRANSFER MODE = 012 running 11-29 TRANSFER MODE = 102 running 11-29 TRANSFER MODE = 112 running 11-31 TRANSFER MODE field descriptions 11-28 table 11-12 transfer rate sync mode 11-55 trap flow figure 7-24 TRAP instruction 7-25 trap vector table (TVT) boot loader use 10-9 to 10-10 definition A-7 trap vector table pointer (TVTP) definition A-7 TRAPcond instruction 7-12, 14-267 traps 7-12, 7-24 initialization 7-24 operation 7-24 overlapping the TVT and IVT 7-25 vector table 7-25 TSTB instruction 14-268 TSTB3 instruction 14-270 TVTP 3-17 See also trap vector table (TVT)

TVTP register 2-10 two parallel stores figure 8-22 two-operand instruction word 8-20 two-operand instructions table 14-4 type one synchronizer maximum delay 12-26 minimum delay 12-26 type three synchronizer maximum delay 12-28 minimum delay 12-27 type two synchronizer maximum delay 12-27 minimum delay 12-27 TYPEx bit 3-14

U

UF flag 3-5 underflow 5-23 unified mode definition A-8 unified mode (DMA) 11-19, 11-35 unsigned-integer formats 5-3

V

V flag 3-5 value in a floating-point number equation 5-4 vector locations table 7-35 vectors (reset, interrupts) 7-28

W

wait state definition A-8
wait states 9-14, 9-36, 9-37 bus disabled 9-38
wait-state generator definition A-8
word transfer operation 12-22, 12-23
word transfer sequence table 12-24
word transfers 11-5
write to AR 8-8

X

XOR instruction 14-272 XOR3 instruction 14-274 XOR3||STI instruction 14-276



Z flag 3-5 zero fill definition A-8

BUSINESS REPLY MAIL FIRST-CLASS MAIL PERMIT NO. 6189 HOUSTON, TX

POSTAGE WILL BE PAID BY ADDRESSEE



P. O. BOX 1443 HOUSTON, TX 77251–9879

Յհակահդիկոսվիկորհիունիունի

NO POSTAG NECESSARY IF MAILED IN THE UNITED STAT

BIND THIS EDGE

Chapter Title—Attribute Reference A-19

PERF

PERF

Reader Response Card: IM5320C4x User's Guid	e	
Please respond to a few questions to help us provide you with the best documentation possible.		Which topics were difficult to fin and why (for example, the top wasn't not in a logical location)?
What is your primary use for the in- formation in this manual?		
Designing 'C4x-based hardware		
Designing 'C4x-based software		
How have you used this manual?		What any other suggestions do yo
To look up specific information or procedures when needed (as a reference)		have for improving this book?
To read chapters about subjects of interest		
To read from front to back before using the information		
Have you found any mistakes or un- clear information in this manual		Name
(please describe and include page		Title
numbers)?		Company
		Address
		City
		State
		Zip/Country
Which topics should be described in greater detail?		
		Can we call you to collect further i formation for improving our doc mentation?
	Thank you.	March, 199

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1996, Texas Instruments Incorporated