TMS320C5x User's Guide

Literature Number: SPRU056D
June 1998

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.

Preface

Read This First

About This Manual

This user's guide describes the architecture, hardware, assembly language instructions, and general operation of the TMS320C5x digital signal processors (DSPs). This manual can also be used as a reference guide for developing hardware and/or software applications.

How to Use This Manual

The following table summarizes the 'C5x information contained in this user's guide:

If you are looking for information about:	Turn to:
Addressing modes	Chapter 5, *Addressing Modes*
Assembly language instructions	Chapter 6, *Assembly Language Instructions*
Boot loader	Chapter 8, *Memory*
Clock generator	Chapter 9, *On-Chip Peripherals*
Control bits	Chapter 4, *Program Control*
CPU	Chapter 3, *Central Processing Unit (CPU)*
Custom ROM from TI	Appendix F, *Submitting ROM Codes to TI*
Development support information	Appendix G, *Development Support and Part Order Information*
Features	Chapter 1, *Introduction* Chapter 2, *Architectural Overview*
Host port interface	Chapter 9, *On-Chip Peripherals*
Input/output ports	Chapter 8, *Memory*
Interrupts	Chapter 4, *Program Control*
Memory configuration	Chapter 8, *Memory*
Memory interface	Chapter 8, *Memory*
On-chip peripherals	Chapter 9, *On-Chip Peripherals*
Opcodes	Chapter 6, *Assembly Language Instructions*
Part order information	Appendix G, *Development Support and Part Order Information*

If you are looking for information about:	Turn to:
Pinouts	Appendix A, *Pinouts and Signal Descriptions*
Pipeline operation	Chapter 7, *Pipeline*
Program control	Chapter 4, *Program Control*
Serial ports	Chapter 9, *On-Chip Peripherals*
Status registers	Chapter 4, *Program Control*
Timer	Chapter 9, *On-Chip Peripherals*
Upgrading from a ’C25	Appendix C, *System Migration*
Wait-state generators	Chapter 9, *On-Chip Peripherals*
XDS510 Emulator	Appendix D, *Design Considerations for Using XDS510 Emulator*

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter’s. Examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

  Here is a sample program listing:

```
0011  0005  0001         .field    1, 2
0012  0005  0003         .field    3, 4
0013  0005  0006         .field    6, 3
0014  0006               .even
```

  Here is an example of a system prompt and a command that you might enter:

```
C:  csr –a /user/ti/simuboard/utilities
```

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered. Here is an example of a directive syntax:

  .asect "*section name***",** *address*

  .asect is the directive. This directive has two parameters, indicated by *section name* and *address*. When you use .asect, the first parameter must be

an actual section name, enclosed in double quotes; the second parameter must be an address.

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

  LALK *16–bit constant [, shift]*

  The LALK instruction has two parameters. The first parameter, *16-bit constant*, is required. The second parameter, *shift*, is optional. As this syntax shows, if you use the optional second parameter, you must precede it with a comma.

  Square brackets are also used as part of the pathname specification for VMS pathnames; in this case, the brackets are actually part of the pathname (they are not optional).

- Braces ({ and }) indicate a list. The symbol **|** (read as *or*) separates items within the list. Here's an example of a list:

  ```
  { * | *+ | *– }
  ```

  This provides three choices: *, *+, or *–.

  Unless the list is enclosed in square brackets, you must choose one item from the list.

- Some directives can have a varying number of parameters. For example, the .byte directive can have up to 100 parameters. The syntax for this directive is:

  .byte $value_1$ *[, ... ,* $value_n$*]*

  This syntax shows that .byte must have at least one value parameter, but you have the option of supplying additional value parameters, separated by commas.

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.

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

Related Documentation From Texas Instruments

The following books describe the ’C5x 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.

TMS320C5x General-Purpose Applications User’s Guide (literature number SPRU164) serves as a reference book for developing hardware and/or software applications for the ’C5x generation of devices.

TMS320C5x, TMS320LC5x Digital Signal Processors (literature number SPRS030) data sheet contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.

TMS320C1x/C2x/C2xx/C5x Code Generation Tools Getting Started Guide (literature number SPRU121) describes how to install the TMS320C1x, TMS320C2x, TMS320C2xx, and TMS320C5x assembly language tools and the C compiler for the ’C1x, ’C2x, ’C2xx, and ’C5x devices. The installation for MS-DOS™, OS/2™, SunOS™, and Solaris™ systems is covered.

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User’s Guide (literature number SPRU018) 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 ’C1x, ’C2x, ’C2xx, and ’C5x generations of devices.

TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide (literature number SPRU024) describes the 'C2x/C2xx/C5x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C5x C Source Debugger User's Guide (literature number SPRU055) tells you how to invoke the 'C5x emulator, evaluation module, 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.

TMS320C5x Evaluation Module Technical Reference (literature number SPRU087) describes the 'C5x evaluation module, its features, design details and external interfaces.

TMS320C5x Evaluation Module Getting Started Guide (literature number SPRU126) tells you how to install the MS-DOS™, PC-DOS™, and Windows™ versions of the 'C5x evaluation module.

TMS320C54x Simulator Getting Started Guide (literature number SPRU137) describes how to install the TMS320C54x simulator and the C source debugger for the 'C54x. The installation for Windows 3.1, SunOS™, and HP-UX™ systems is covered.

XDS51x Emulator Installation Guide (literature number SPNU070) describes the installation of the XDS510™, XDS510PP™, and XDS510WS™ emulator controllers. The installation of the XDS511™ emulator is also described.

JTAG/MPSD Emulation Technical Reference (literature number SPDU079) provides the design requirements of the XDS510™ emulator controller, discusses JTAG designs (based on the IEEE 1149.1 standard), and modular port scan device (MPSD) designs.

TMS320 Third-Party Support Reference Guide (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of TMS320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

TMS320 DSP Development Support Reference Guide (literature number SPRU011) describes the TMS320 family of digital signal processors and the tools that support these devices. Included are code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). Also covered are available documentation, seminars, the university program, and factory repair and exchange.

If you are an assembly language programmer and would like more information about C or C expressions, you may find this book useful:

The C Programming Language (second edition, 1988), by Brian W. Kernighan and Dennis M. Ritchie, published by Prentice-Hall, Englewood Cliffs, New Jersey.

Technical Articles

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.

3) Burrus, C.S., and T.W. Parks, *DFT/FFT and Convolution Algorithms*, New York, NY: John Wiley and Sons, Inc., 1984.

4) 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.

5) Defatta, David J., Joseph G. Lucas, and William S. Hodgkiss, *Digital Signal Processing: A System Design Approach*, New York: John Wiley, 1988.

6) 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.

7) 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.

8) 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.

9) 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.

10) Gold, Bernard, and C.M. Rader, *Digital Processing of Signals*, New York, NY: McGraw-Hill Company, Inc., 1969.

11) 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.

14) 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.

22) Oppenheim, A.V., A.N. Willsky, and I.T. Young, *Signals and Systems*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

23) 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.

24) 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.

27) 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.

2) Gonzales, Rafael C., and Paul Wintz, *Digital Image Processing*, Reading, MA: Addison-Wesley Publishing Company, Inc., 1977.

3) Papamichalis, P.E., "FFT Implementation on the TMS320C30." *Proceedings of ICASSP 88*, USA, Volume D, page 1399, April 1988.

4) 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**:*

1) 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.

2) 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.

3) 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.

5) Papamichalis, Panos, *Practical Approaches to Speech Coding*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

6) 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.

7) 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.

8) Rabiner, L.R., and R.W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.

9) Reimer, J.B. and K.S. Lin, "TMS320 Digital Signal Processors in Speech Applications." *Proceedings of SPEECH TECH '88*, April 1988.

10) 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**:*

1) Ahmed, I., "16-Bit DSP Microcontroller Fits Motion Control System Application." *PCIM*, October 1988.

2) Ahmed, I., "Implementation of Self Tuning Regulators with TMS320 Family of Digital Signal Processors." *MOTORCON '88*, pages 248–262, September 1988.

3) Ahmed, I., and S. Lindquist, "Digital Signal Processors: Simplifying High-Performance Control." *Machine Design*, September 1987.

4) Ahmed, I., and S. Meshkat, "Using DSPs in Control." *Control Engineering*, February 1988.

5) 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.

6) 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.

7) 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.

9) 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.

11) Lovrich, A., G. Troullinos, and R. Chirayil, "An All-Digital Automatic Gain Control." *Proceedings of ICASSP 88*, USA, Volume D, page 1734, April 1988.

12) 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.

13) 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.

2) Reimer, J., G. Benbassat, and W. Bonneau Jr., “Application Processors: Making PC Multimedia Happen.” *Silicon Valley PC Design Conference*, July 1991.

***Military**:*

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

***Telecommunications**:*

1) Ahmed, I., and A. Lovrich, “Adaptive Line Enhancer Using the TMS320C25.” *Conference Records of Northcon/86*, USA, 14/3/1–10, September/October 1986.

2) 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.

3) 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.

6) Lovrich, A. and J. Reimer, “A Multi-Rate Transcoder.” *Digest of Technical Papers for 1989 International Conference on Consumer Electronics*, June 7–9, 1989.

7) 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.

8) 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.

10) 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**:*

1) 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.

2) Reimer, J.B., and G.A. Frantz, “Customization of a DSP Integrated Circuit for a Customer Product.” *Transactions on Consumer Electronics*, USA, August 1988.

3) 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**:*

1) 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.

2) 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.

Trademarks

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324	Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323		

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32

Email: epic@ti.com

Deutsch	+49 8161 80 33 11 or	+33 1 30 70 11 68
English		+33 1 30 70 11 65
Francais		+33 1 30 70 11 64
Italiano		+33 1 30 70 11 67
EPIC Modem BBS		+33 1 30 70 11 99
European Factory Repair		+33 4 93 22 25 40
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

Figures

Tables

Examples

Chapter 1

Introduction

This user's guide discusses the TMS320C5x generation of fixed-point digital signal processors (DSPs) in the TMS320 family. The 'C5x DSP provides improved performance over earlier 'C1x and 'C2x generations while maintaining upward compatibility of source code between the devices. The 'C5x central processing unit (CPU) is based on the 'C25 CPU and incorporates additional architectural enhancements that allow the device to run twice as fast as 'C2x devices. Future expansion and enhancements are expected to heighten the performance and range of applications of the 'C5x DSPs.

The 'C5x generation of static CMOS DSPs consists of the following devices:

Device	On-Chip RAM	On-Chip ROM
TMS320C50/LC50	10K words	2K words
TMS320C51/LC51	2K words	8K words
TMS320C52/LC52	1K words	4K words
TMS320C53/LC53	4K words	16K words
TMS320C53S/LC53S	4K words	16K words
TMS320LC56	7K words	32K words
TMS320LC57	7K words	32K words
TMS320C57S/LC57S	7K words	2K words

Topic **Page**

1.1 TMS320 Family Overview

The TMS320 family consists of two types of single-chip DSPs: 16-bit fixed-point and 32-bit floating-point. These DSPs possess the operational flexibility of high-speed controllers and the numerical capability of array processors. Combining these two qualities, the TMS320 processors are inexpensive alternatives to custom-fabricated VLSI and multichip bit-slice processors. Refer to subsection 1.1.2, *TMS320 Typical Applications*, for a detailed list of applications of the TMS320 family. The following characteristics make this family the ideal choice for a wide range of processing applications:

- ❑ Very flexible instruction set
- ❑ Inherent operational flexibility
- ❑ High-speed performance
- ❑ Innovative, parallel architectural design
- ❑ Cost-effectiveness

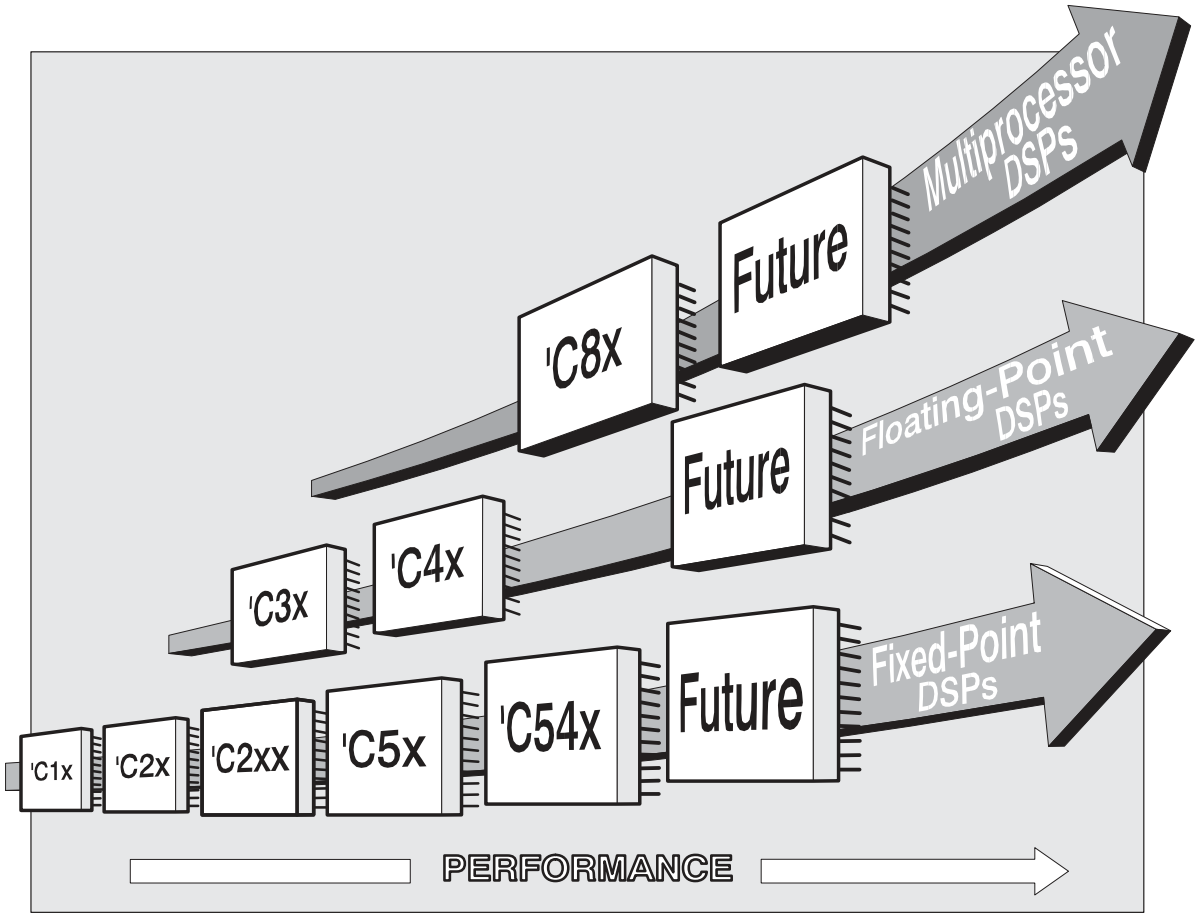
1.1.1 History, Development, and Advantages of TMS320 DSPs

In 1982, Texas Instruments introduced the TMS32010 — the first fixed-point DSP in the TMS320 family. Before the end of the year, the *Electronic Products* magazine awarded the TMS32010 the title “Product of the Year”. The TMS32010 became the model for future TMS320 generations.

Today, the TMS320 family consists of eight generations: the ’C1x, ’C2x, ’C2xx, ’C5x, and ’C54x are fixed-point, the ’C3x and ’C4x are floating-point, and the ’C8x is a multiprocessor. Figure 1–1 illustrates the performance gains that the TMS320 family has made over time with successive generations. Source code is upward compatible from one fixed-point generation to the next fixed-point generation (except for the ’C54x), and from one floating-point generation to the next floating-point generation. Upward compatibility preserves the software generation of your investment, thereby providing a convenient and cost-efficient means to a higher-performance, more versatile DSP system.

Each generation of TMS320 devices has a CPU and a variety of on-chip memory and peripheral configurations for developing spin-off devices. These spin-off devices satisfy a wide range of needs in the worldwide electronics market. When memory and peripherals are integrated into one processor, the overall system cost is greatly reduced, and circuit board space is saved.

Figure 1–1. Evolution of the TMS320 Family



1.1.2 TMS320 Typical Applications

The TMS320 family of DSPs offers better, more adaptable approaches to traditional signal-processing problems, such as vocoding, filtering, and error coding. Furthermore, the TMS320 family supports complex applications that often require multiple operations to be performed simultaneously. Figure 1–2 shows many of the typical applications of the TMS320 family.

Figure 1–2. Typical Applications for the TMS320 Family

Automotive	**Consumer**	**Control**
Adaptive ride control	Digital radios/TVs	Disk drive control
Antiskid brakes	Educational toys	Engine control
Cellular telephones	Music synthesizers	Laser printer control
Digital radios	Power tools	Motor control
Engine control	Radar detectors	Robotics control
Global positioning	Solid-state answering machines	Servo control
Navigation		
Vibration analysis		
Voice commands		
General-Purpose	**Graphics/Imaging**	**Industrial**
Adaptive filtering	3-D rotation	Numeric control
Convolution	Animation/digital map	Power-line monitoring
Correlation	Homomorphic processing	Robotics
Digital filtering	Pattern recognition	Security access
Fast Fourier transforms	Image enhancement	
Hilbert transforms	Image compression/transmission	
Waveform generation	Robot vision	
Windowing	Workstations	
Instrumentation	**Medical**	**Military**
Digital filtering	Diagnostic equipment	Image processing
Function generation	Fetal monitoring	Missile guidance
Pattern matching	Hearing aids	Navigation
Phase-locked loops	Patient monitoring	Radar processing
Seismic processing	Prosthetics	Radio frequency modems
Spectrum analysis	Ultrasound equipment	Secure communications
Transient analysis		Sonar processing
Telecommunications		**Voice/Speech**
1200- to 19200-bps modems	DTMF encoding/decoding	Speech enhancement
Adaptive equalizers	Echo cancellation	Speech recognition
ADPCM transcoders	Fax	Speech synthesis
Cellular telephones	Line repeaters	Speaker verification
Channel multiplexing	Speaker phones	Speech vocoding
Data encryption	Spread spectrum communications	Voice mail
Digital PBXs	Video conferencing	Text-to-speech
Digital speech interpolation (DSI)	X.25 Packet Switching	
Personal digital assistants (PDA)	Personal communications systems (PCS)	

1.2 TMS320C5x Overview

The ’C5x generation consists of the ’C50, ’C51, ’C52, ’C53, ’C53S, ’C56, ’C57, and ’C57S DSPs, which are fabricated by CMOS integrated-circuit technology. Their architectural design is based on the ’C25. The operational flexibility and speed of the ’C5x are the result of combining an advanced Harvard architecture (which has separate buses for program memory and data memory), a CPU with application-specific hardware logic, on-chip peripherals, on-chip memory, and a highly specialized instruction set. The ’C5x is designed to execute up to 50 million instructions per second (MIPS). Spin-off devices that combine the ’C5x CPU with customized on-chip memory and peripheral configurations may be developed for special applications in the worldwide electronics market.

The ’C5x devices offer these advantages:

- ❏ Enhanced TMS320 architectural design for increased performance and versatility
- ❏ Modular architectural design for fast development of spin-off devices
- ❏ Advanced integrated-circuit processing technology for increased performance and low power consumption
- ❏ Source code compatibility with ’C1x, ’C2x, and ’C2xx DSPs for fast and easy performance upgrades
- ❏ Enhanced instruction set for faster algorithms and for optimized high-level language operation
- ❏ Reduced power consumption and increased radiation hardness because of new static design techniques

Table 1–1 lists the major characteristics of the ’C5x DSPs. The table shows the capacity of on-chip RAM and ROM, number of serial and parallel input/output (I/O) ports, power supply requirements, execution time of one machine cycle, and package types available with total pin count. Use Table 1–1 for guidance in choosing the best ’C5x DSP for your application.

Table 1–1. Characteristics of the ’C5x DSPs

TMS320 Device	ID	On-Chip Memory (16-bit words) DARAM†	SARAM‡	ROM	I/O Ports Serial	Parallel ◊	Power Supply (V)	Cycle Time (ns)	Package Type
’C50	PQ	1056	9K	2K§	2¶	64K	5	50/35/25	132 pin BQFP○
’LC50	PQ	1056	9K	2K§	2¶	64K	3.3	50/40/25	132 pin BQFP○
’C51	PQ	1056	1K	8K§	2¶	64K	5	50/35/25/20	132 pin BQFP○
’C51	PZ	1056	1K	8K§	2¶	64K	5	50/35/25/20	100 pin TQFP☆
’LC51	PQ	1056	1K	8K§	2¶	64K	3.3	50/40/25	132 pin BQFP○
’LC51	PZ	1056	1K	8K§	2¶	64K	3.3	50/40/25	100 pin TQFP☆
’C52	PJ	1056	—	4K§	1	64K	5	50/35/25/20	100 pin QFP□
’C52	PZ	1056	—	4K§	1	64K	5	50/35/25/20	100 pin TQFP☆
’LC52	PJ	1056	—	4K§	1	64K	3.3	50/40/25	100 pin QFP□
’LC52	PZ	1056	—	4K§	1	64K	3.3	50/40/25	100 pin TQFP☆
’C53	PQ	1056	3K	16K§	2¶	64K	5	50/35/25	132 pin BQFP○
’C53S	PZ	1056	3K	16K§	2	64K	5	50/35/25	100 pin TQFP☆
’LC53	PQ	1056	3K	16K§	2¶	64K	3.3	50/40/25	132 pin BQFP○
’LC53S	PZ	1056	3K	16K§	2	64K	3.3	50/40/25	100 pin TQFP☆
’LC56	PZ	1056	6K	32K	2#	64K	3.3	50/35/25	100 pin TQFP☆
’C57S	PGE	1056	6K	2K§	2#	64K‖	5	50/35/25	144 pin TQFPΔ
’LC57	PBK	1056	6K	32K	2#	64K‖	3.3	50/35/25	128 pinTQFP☆
’LC57S	PGE	1056	6K	2K§	2#	64K‖	3.3	50/35	144 pin TQFPΔ

† Dual-access RAM (DARAM)
‡ Single-access RAM (SARAM)
§ ROM bootloader available
¶ Includes time-division multiplexed (TDM) serial port
# Includes buffered serial port (BSP)
‖ Includes host port interface (HPI)
○ $20 \times 20 \times 3.8$ mm bumpered quad flat-pack (BQFP) package
☆ $14 \times 14 \times 1.4$ mm thin quad flat-pack (TQFP) package
□ $14 \times 20 \times 2.7$ mm quad flat-pack (QFP) package
Δ $20 \times 20 \times 1.4$ mm thin quad flat-pack (TQFP) package
◊ Sixteen of the 64K parallel I/O ports are memory mapped.

1.3 TMS320C5x Key Features

Key features of the ’C5x DSPs are listed below. Where a feature is exclusive to a particular device, the device’s name is enclosed within parentheses and noted after that feature.

- Compatibility: Source-code compatible with ’C1x, ’C2x, and ’C2xx devices
- Speed: 20-/25-/35-/50-ns single-cycle fixed-point instruction execution time (50/40/28.6/20 MIPS)
- Power
  - 3.3-V and 5-V static CMOS technology with two power-down modes
  - Power consumption control with IDLE1 and IDLE2 instructions for power-down modes
- Memory
  - 224K-word × 16-bit maximum addressable external memory space (64K-word program, 64K-word data, 64K-word I/O, and 32K-word global memory)
  - 1056-word × 16-bit dual-access on-chip data RAM
  - 9K-word × 16-bit single-access on-chip program/data RAM (’C50)
  - 2K-word × 16-bit single-access on-chip boot ROM (’C50, ’C57S)
  - 1K-word × 16-bit single-access on-chip program/data RAM (’C51)
  - 8K-word × 16-bit single-access on-chip program ROM (’C51)
  - 4K-word × 16-bit single-access on-chip program ROM (’C52)
  - 3K-word × 16-bit single-access on-chip program/data RAM (’C53, ’C53S)
  - 16K-word × 16-bit single-access on-chip program ROM (’C53, ’C53S)
  - 6K-word × 16-bit single-access on-chip program/data RAM (’LC56, ’C57S, ’LC57)
  - 32K-word × 16-bit single-access on-chip program ROM (’LC56, ’LC57)

- Central processing unit (CPU)
  - Central arithmetic logic unit (CALU) consisting of the following:
    - 32-bit arithmetic logic unit (ALU), 32-bit accumulator (ACC), and 32-bit accumulator buffer (ACCB)
    - 16-bit × 16-bit parallel multiplier with a 32-bit product capability
    - 0- to 16-bit left and right data barrel-shifters and a 64-bit incremental data shifter
  - 16-bit parallel logic unit (PLU)
  - Dedicated auxiliary register arithmetic unit (ARAU) for indirect addressing
  - Eight auxiliary registers
- Program control
  - 8-level hardware stack
  - 4-deep pipelined operation for delayed branch, call, and return instructions
  - Eleven shadow registers for storing strategic CPU-controlled registers during an interrupt service routine (ISR)
  - Extended hold operation for concurrent external direct memory access (DMA) of external memory or on-chip RAM
  - Two indirectly addressed circular buffers for circular addressing
- Instruction set
  - Single-cycle multiply/accumulate instructions
  - Single-instruction repeat and block repeat operations
  - Block memory move instructions for better program and data management
  - Memory-mapped register load and store instructions
  - Conditional branch and call instructions
  - Delayed execution of branch and call instructions
  - Fast return from interrupt instructions
  - Index-addressing mode
  - Bit-reversed index-addressing mode for radix-2 fast-Fourier transforms (FFTs)

- On-chip peripherals
  - 64K parallel I/O ports (16 I/O ports are memory-mapped)
  - Sixteen software-programmable wait-state generators for program, data, and I/O memory spaces
  - Interval timer with period, control, and counter registers for software stop, start, and reset
  - Phase-locked loop (PLL) clock generator with internal oscillator or external clock source
  - Multiple PLL clocking option (x1, x2, x3, x4, x5, x9, depending on the device)
  - Full-duplex synchronous serial port interface for direct communication between the ’C5x and another serial device
  - Time-division multiplexed (TDM) serial port (’C50, ’C51, ’C53)
  - Buffered serial port (BSP) (’LC56, ’C57S, ’LC57)
  - 8-bit parallel host port interface (HPI) (’C57, ’C57S)
- Test/Emulation
  - On-chip scan-based emulation logic
  - IEEE JTAG Standard 1149.1 boundary scan logic (’C50, ’C51, ’C53, ’C57S)
- Packages
  - 100-pin quad flat-pack (QFP) package (’C52)
  - 100-pin thin quad flat-pack (TQFP) package (’C51, ’C52, ’C53S, ’LC56)
  - 128-pin TQFP package (’LC57)
  - 132-pin bumpered quad flat-pack (BQFP) package (’C50, ’C51, ’C53)
  - 144-pin TQFP package (’C57S)

Chapter 2

Architectural Overview

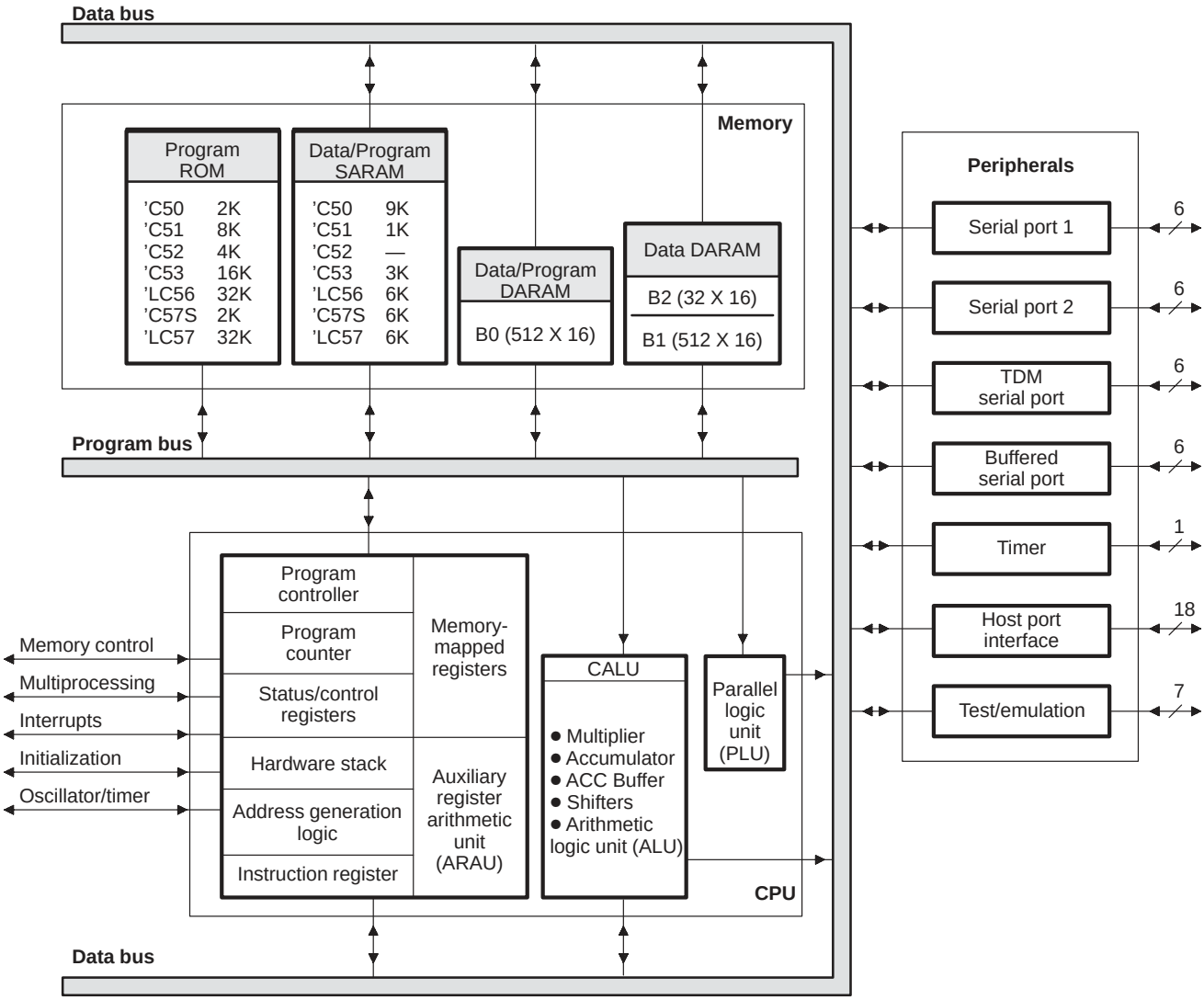
This chapter provides an overview of the architectural structure of the ’C5x, which consists of the buses, on-chip memory, central processing unit (CPU), and on-chip peripherals.

The ’C5x uses an advanced, modified Harvard-type architecture based on the ’C25 architecture and maximizes processing power with separate buses for program memory and data memory. The instruction set supports data transfers between the two memory spaces. Figure 2–1 shows a functional block diagram of the ’C5x.

All ’C5x DSPs have the same CPU structure; however, they have different on-chip memory configurations and on-chip peripherals.

Topic **Page**

Figure 2–1. 'C5x Functional Block Diagram



2.1 Bus Structure

Separate program and data buses allow simultaneous access to program instructions and data, providing a high degree of parallelism. For example, while data is multiplied, a previous product can be loaded into, added to, or subtracted from the accumulator and, at the same time, a new address can be generated. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. In addition, the ’C5x includes the control mechanisms to manage interrupts, repeated operations, and function calling.

The ’C5x architecture is built around four major buses:

- Program bus (PB)
- Program address bus (PAB)
- Data read bus (DB)
- Data read address bus (DAB)

The PAB provides addresses to program memory space for both reads and writes. The PB also carries the instruction code and immediate operands from program memory space to the CPU. The DB interconnects various elements of the CPU to data memory space. The program and data buses can work together to transfer data from on-chip data memory and internal or external program memory to the multiplier for single-cycle multiply/accumulate operations.

2.2 Central Processing Unit (CPU)

The ’C5x CPU consists of these elements:

- ❑ Central arithmetic logic unit (CALU)
- ❑ Parallel logic unit (PLU)
- ❑ Auxiliary register arithmetic unit (ARAU)
- ❑ Memory-mapped registers
- ❑ Program controller

The ’C5x CPU maintains source-code compatibility with the ’C1x and ’C2x generations while achieving high performance and greater versatility. Improvements include a 32-bit accumulator buffer, additional scaling capabilities, and a host of new instructions. The instruction set exploits the additional hardware features and is flexible in a wide range of applications. Data management has been improved through the use of new block move instructions and memory-mapped register instructions. See Chapter 3, *Central Processing Unit (CPU)*.

2.2.1 Central Arithmetic Logic Unit (CALU)

The CPU uses the CALU to perform 2s-complement arithmetic. The CALU consists of these elements:

- ❑ 16-bit $\times$ 16-bit multiplier
- ❑ 32-bit arithmetic logic unit (ALU)
- ❑ 32-bit accumulator (ACC)
- ❑ 32-bit accumulator buffer (ACCB)
- ❑ Additional shifters at the outputs of both the accumulator and the product register (PREG)

For information on the CALU, see Section 3.2, *Central Arithmetic Logic Unit (CALU)*, on page 3-7.

2.2.2 Parallel Logic Unit (PLU)

The CPU includes an independent PLU, which operates separately from, but in parallel with, the ALU. The PLU performs Boolean operations or the bit manipulations required of high-speed controllers. The PLU can set, clear, test, or toggle bits in a status register, control register, or any data memory location. The PLU provides a direct logic operation path to data memory values without affecting the contents of the ACC or PREG. Results of a PLU function are written back to the original data memory location. For information on the PLU, see Section 3.3, *Parallel Logic Unit (PLU)*, on page 3-15.

2.2.3 Auxiliary Register Arithmetic Unit (ARAU)

The CPU includes an unsigned 16-bit arithmetic logic unit that calculates indirect addresses by using inputs from the auxiliary registers (ARs), index register (INDX), and auxiliary register compare register (ARCR). The ARAU can autoindex the current AR while the data memory location is being addressed and can index either by ± 1 or by the contents of the INDX. As a result, accessing data does not require the CALU for address manipulation; therefore, the CALU is free for other operations in parallel. For information on the ARAU, see Section 3.4, *Auxiliary Register Arithmetic Unit (ARAU)*, on page 3-17.

2.2.4 Memory-Mapped Registers

The ’C5x has 96 registers mapped into page 0 of the data memory space. All ’C5x DSPs have 28 CPU registers and 16 input/output (I/O) port registers but have different numbers of peripheral and reserved registers (see Chapter 4, *Memory*). Since the memory-mapped registers are a component of the data memory space, they can be written to and read from in the same way as any other data memory location. The memory-mapped registers are used for indirect data address pointers, temporary storage, CPU status and control, or integer arithmetic processing through the ARAU. For information on registers, see Section 3.5, *Summary of Registers*, on page 3-21.

2.2.5 Program Controller

The program controller contains logic circuitry that decodes the operational instructions, manages the CPU pipeline, stores the status of CPU operations, and decodes the conditional operations. Parallelism of architecture lets the ’C5x perform three concurrent memory operations in any given machine cycle: fetch an instruction, read an operand, and write an operand. See Chapter 4, *Program Control*, and Chapter 7, *Pipeline*. The program controller consists of these elements:

- ❏ Program counter
- ❏ Status and control registers
- ❏ Hardware stack
- ❏ Address generation logic
- ❏ Instruction register

2.3 On-Chip Memory

The ’C5x architecture contains a considerable amount of on-chip memory to aid in system performance and integration:

- ❑ Program read-only memory (ROM)
- ❑ Data/program dual-access RAM (DARAM)
- ❑ Data/program single-access RAM (SARAM)

The ’C5x has a total address range of 224K words × 16 bits. The memory space is divided into four individually selectable memory segments: 64K-word program memory space, 64K-word local data memory space, 64K-word input/output ports, and 32K-word global data memory space. For information on the memory organization, see Chapter 8, *Memory*.

2.3.1 Program ROM

All ’C5x DSPs carry a 16-bit on-chip maskable programmable ROM (see Table 1–1 for sizes). The ’C50 and ’C57S DSPs have boot loader code resident in the on-chip ROM, all other ’C5x DSPs offer the boot loader code as an option. This memory is used for booting program code from slower external ROM or EPROM to fast on-chip or external RAM. Once the custom program has been booted into RAM, the boot ROM space can be removed from program memory space by setting the MP/$\overline{\text{MC}}$ bit in the processor mode status register (PMST). The on-chip ROM is selected at reset by driving the MP/$\overline{\text{MC}}$ pin low. If the on-chip ROM is not selected, the ’C5x devices start execution from off-chip memory. For information on the program ROM, see Section 8.2, *Program Memory*, on page 8-7.

The on-chip ROM may be configured with or without boot loader code. However, the on-chip ROM is intended for your specific program. Once the program is in its final form, you can submit the ROM code to Texas Instruments for implementation into your device. For details on how to submit code to Texas Instruments to program your ROM, see Appendix F, *Submitting ROM Codes to TI*.

2.3.2 Data/Program Dual-Access RAM

All ’C5x DSPs carry a 1056-word × 16-bit on-chip dual-access RAM (DARAM). The DARAM is divided into three individually selectable memory blocks: 512-word data or program DARAM block B0, 512-word data DARAM block B1, and 32-word data DARAM block B2. The DARAM is primarily intended to store data values but, when needed, can be used to store programs as well. DARAM blocks B1 and B2 are always configured as data memory; however, DARAM

block B0 can be configured by software as data or program memory. The DARAM can be configured in one of two ways:

- ❑ All 1056 words × 16 bits configured as data memory
- ❑ 544 words × 16 bits configured as data memory and 512 words × 16 bits configured as program memory

DARAM improves the operational speed of the ’C5x CPU. The CPU operates with a 4-deep pipeline. In this pipeline, the CPU reads data on the third stage and writes data on the fourth stage. Hence, for a given instruction sequence, the second instruction could be reading data at the same time the first instruction is writing data. The dual data buses (DB and DAB) allow the CPU to read from and write to DARAM in the same machine cycle. For information on DARAM, see Section 8.3, *Local Data Memory*, on page 8-15.

2.3.3 Data/Program Single-Access RAM

All ’C5x DSPs except the ’C52 carry a 16-bit on-chip single-access RAM (SARAM) of various sizes (see Table 1–1). Code can be booted from an off-chip ROM and then executed at full speed, once it is loaded into the on-chip SARAM. The SARAM can be configured by software in one of three ways:

- ❑ All SARAM configured as data memory
- ❑ All SARAM configured as program memory
- ❑ SARAM configured as both data memory and program memory

The SARAM is divided into 1K- and/or 2K-word blocks contiguous in address memory space. All ’C5x CPUs support parallel accesses to these SARAM blocks. However, one SARAM block can be accessed only once per machine cycle. In other words, the CPU can read from or write to one SARAM block while accessing another SARAM block. When the CPU requests multiple accesses, the SARAM schedules the accesses by providing a not-ready condition to the CPU and executing the multiple accesses one cycle at a time.

SARAM supports more flexible address mapping than DARAM because SARAM can be mapped to both program and data memory space simultaneously. However, because of simultaneous program and data mapping, an instruction fetch and data fetch that could be performed in one machine cycle with DARAM may take two machine cycles with SARAM. For information on SARAM, see Section 8.3, *Local Data Memory*, on page 8-15.

2.3.4 On-Chip Memory Protection

The ’C5x DSPs have a maskable option that protects the contents of on-chip memories. When the related bit is set, no externally originating instruction can access the on-chip memory spaces. For information on the protection feature, see subsection 8.2.4, *Program Memory Protection Feature*, on page 8-14.

2.4 On-Chip Peripherals

All ’C5x DSPs have the same CPU structure; however, they have different on-chip peripherals connected to their CPUs. The ’C5x DSP on-chip peripherals available are:

- ❑ Clock generator
- ❑ Hardware timer
- ❑ Software-programmable wait-state generators
- ❑ Parallel I/O ports
- ❑ Host port interface (HPI)
- ❑ Serial port
- ❑ Buffered serial port (BSP)
- ❑ Time-division multiplexed (TDM) serial port
- ❑ User-maskable interrupts

2.4.1 Clock Generator

The clock generator consists of an internal oscillator and a phase-locked loop (PLL) circuit. The clock generator can be driven internally by a crystal resonator circuit or driven externally by a clock source. The PLL circuit can generate an internal CPU clock by multiplying the clock source by a specific factor, so you can use a clock source with a lower frequency than that of the CPU. For information, see Section 9.2, *Clock Generator*, on page 9-7.

2.4.2 Hardware Timer

A 16-bit hardware timer with a 4-bit prescaler is available. This programmable timer clocks at a rate that is between 1/2 and 1/32 of the machine cycle rate (CLKOUT1), depending upon the timer’s divide-down ratio. The timer can be stopped, restarted, reset, or disabled by specific status bits. For information, see Section 9.3, *Timer*, on page 9-9.

2.4.3 Software-Programmable Wait-State Generators

Software-programmable wait-state logic is incorporated in ’C5x DSPs allowing wait-state generation without any external hardware for interfacing with slower off-chip memory and I/O devices. This feature consists of multiple wait-state generating circuits. Each circuit is user-programmable to operate in different wait states for off-chip memory accesses. For information, see Section 9.4, *Software-Programmable Wait-State Generators*, on page 9-13.

2.4.4 Parallel I/O Ports

A total of 64K I/O ports are available, sixteen of these ports are memory-mapped in data memory space. Each of the I/O ports can be addressed by the IN or the OUT instruction. The memory-mapped I/O ports can be accessed with any instruction that reads from or writes to data memory. The $\overline{IS}$ signal indicates a read or write operation through an I/O port. The ’C5x can easily interface with external I/O devices through the I/O ports while requiring minimal off-chip address decoding circuits. For information, see Section 9.6, *Parallel I/O Ports*, on page 9-22.

Table 2–1 lists the number and type of parallel ports available in ’C5x DSPs with various package types.

2.4.5 Host Port Interface (HPI)

The HPI available on the ’C57S and ’LC57 is an 8-bit parallel I/O port that provides an interface to a host processor. Information is exchanged between the DSP and the host processor through on-chip memory that is accessible to both the host processor and the ’C57. For information, see Section 9.10, *Host Port Interface*, on page 9-87.

Table 2–1. Number of Serial/Parallel Ports Available in Different ’C5x Package Types

TMS320 Device	Package ID†	High-Speed Serial Port	TDM Serial Port	Buffered Serial Port	Host Port (Parallel)
’C50/’LC50	PQ	1	1	–	–
’C51/’LC51	PQ/PZ	1	1	–	–
’C52/’LC52	PJ/PZ	1	–	–	–
’C53/’LC53	PQ	1	1	–	–
’C53S/’LC53S	PZ	2	–	–	–
’LC56	PZ	1	–	1	–
’C57S/’LC57S	PGE	1	–	1	1
’LC57	PBK	1	–	1	1

† PGE is a 20 × 20 × 1.4 mm thin quad flat-pack (TQFP) package
PJ is a 14 × 20 × 2.7 mm quad flat-pack (QFP) package
PQ is a 20 × 20 × 3.8 mm bumpered quad flat-pack (BQFP) package
PZ and PBK are a 14 × 14 × 1.4 mm thin quad flat-pack (TQFP) package

2.4.6 Serial Port

Three different kinds of serial ports are available: a general-purpose serial port, a time-division multiplexed (TDM) serial port, and a buffered serial port (BSP). Each ’C5x contains at least one general-purpose, high-speed synchronous, full-duplexed serial port interface that provides direct communication with serial devices such as codecs, serial analog-to-digital (A/D) converters, and other serial systems. The serial port is capable of operating at up to one-fourth the machine cycle rate (CLKOUT1). The serial port transmitter and receiver are double-buffered and individually controlled by maskable external interrupt signals. Data is framed either as bytes or as words.

Table 2–1 lists the number and type of serial ports available in ’C5x DSPs with various package types. For information on serial ports, see Section 9.7, *Serial Port Interface*, on page 9-23.

2.4.7 Buffered Serial Port (BSP)

The BSP available on the ’C56 and ’C57 devices is a full-duplexed, double-buffered serial port and an autobuffering unit (ABU). The BSP provides flexibility on the data stream length. The ABU supports high-speed data transfer and reduces interrupt latencies.

Table 2–1 lists the number and type of serial ports available in ’C5x DSPs with various package types. For information, see Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53.

2.4.8 TDM Serial Port

The TDM serial port available on the ’C50, ’C51, and ’C53 devices is a full-duplexed serial port that can be configured by software either for synchronous operations or for time-division multiplexed operations. The TDM serial port is commonly used in multiprocessor applications.

Table 2–1 lists the number and type of serial ports available in ’C5x DSPs with various package types. For information, see Section 9.9, *Time-Division Multiplexed (TDM) Serial Port Interface*, on page 9-74.

2.4.9 User-Maskable Interrupts

Four external interrupt lines ($\overline{\text{INT1}}$–$\overline{\text{INT4}}$) and five internal interrupts, a timer interrupt and four serial port interrupts, are user maskable. When an interrupt service routine (ISR) is executed, the contents of the program counter are saved on an 8-level hardware stack, and the contents of eleven specific CPU registers are automatically saved (shadowed) on a 1-level-deep stack. When a return from interrupt instruction is executed, the CPU registers’ contents are restored. For information, see Section 4.8, *Interrupts*, on page 4-36.

2.5 Test/Emulation

On the ’C50, ’LC50, ’C51, ’LC51, ’C53, ’LC53, ’C57S and ’LC57S, an IEEE standard 1149.1 (JTAG) interface with boundary scan capability is used for emulation and test. This logic provides the boundary scan to and from the interfacing devices. It can be used to test pin-to-pin continuity and to perform operational tests on devices that are peripheral to the ’C5x.

On the ’C52, ’LC52, ’C53S, ’LC53S, ’LC56, and ’LC57, an IEEE standard 1149.1 (JTAG) interface without boundary scan capability is used for emulation purposes only and is interfaced to other internal scanning logic circuitry that has access to all of the on-chip resources. Thus, the ’C5x can perform on-board emulation by means of the IEEE standard 1149.1 serial scan pins and the emulation-dedicated pins.

The on-chip analysis block in conjunction with the ’C5x debugger software provides the capability to perform debugging and performance evaluation functions in a target system. The full analysis block provides the following capabilities:

- Flexible breakpoint setup. Breakpoints can be triggered based on the following events:
  - Program fetches/reads/writes
  - EMU0/1 pin activity
  - Data reads/writes
  - CPU events (calls, returns, interrupts/traps, branches, pipeline clock)
  - Event counter overflow
- Counting of the following events for performance analysis:
  - CPU clocks
  - Pipeline advances
  - Instruction fetches
  - Calls, returns, interrupts/traps, branches
  - Program fetches/reads/writes
  - Data reads/writes
- Program counter discontinuity trace buffer to monitor program counter flow.

The reduced analysis block on the ’C53S and ’LC53S provides the capability for breakpoint triggering based on program fetches/reads/writes and EMU0/1 pin activity.

Table 2–2 lists the IEEE standard 1149.1 (JTAG) interface, boundary scan capability, and on-chip analysis block functions supported by the ’C5x. See IEEE Std. 1149.1 for more details.

Refer to the *TMS320 DSP Development Support Reference Guide* for additional information on available TMS320 development tools.

Table 2–2. IEEE Std.1149.1 (JTAG)/Boundary-Scan Interface Configurations for the ’C5x

TMS320 Device	IEEE Std.1149.1 Interface	Boundary Scan Capability	On-Chip Analysis Block
’C50/’LC50	Yes	Yes	Full
’C51/’LC51	Yes	Yes	Full
’C52/’LC52	Yes	No	Full
’C53/’LC53	Yes	Yes	Full
’C53S/’LC53S	Yes	No	Reduced
’LC56	Yes	No	Full
’C57S/’LC57S	Yes	Yes	Full
’LC57	Yes	No	Full

Chapter 3

Central Processing Unit (CPU)

The TMS320C5x DSP central processing unit (CPU) can perform high-speed arithmetic within a short instruction cycle by means of its highly parallel architecture, which consists of the following elements:

- Program controller
- Central arithmetic logic unit (CALU)
- Parallel logic unit (PLU)
- Auxiliary register arithmetic unit (ARAU)
- Memory-mapped registers

This chapter does not discuss the memory and peripheral segments, except in relation to the CPU.

Topic **Page**

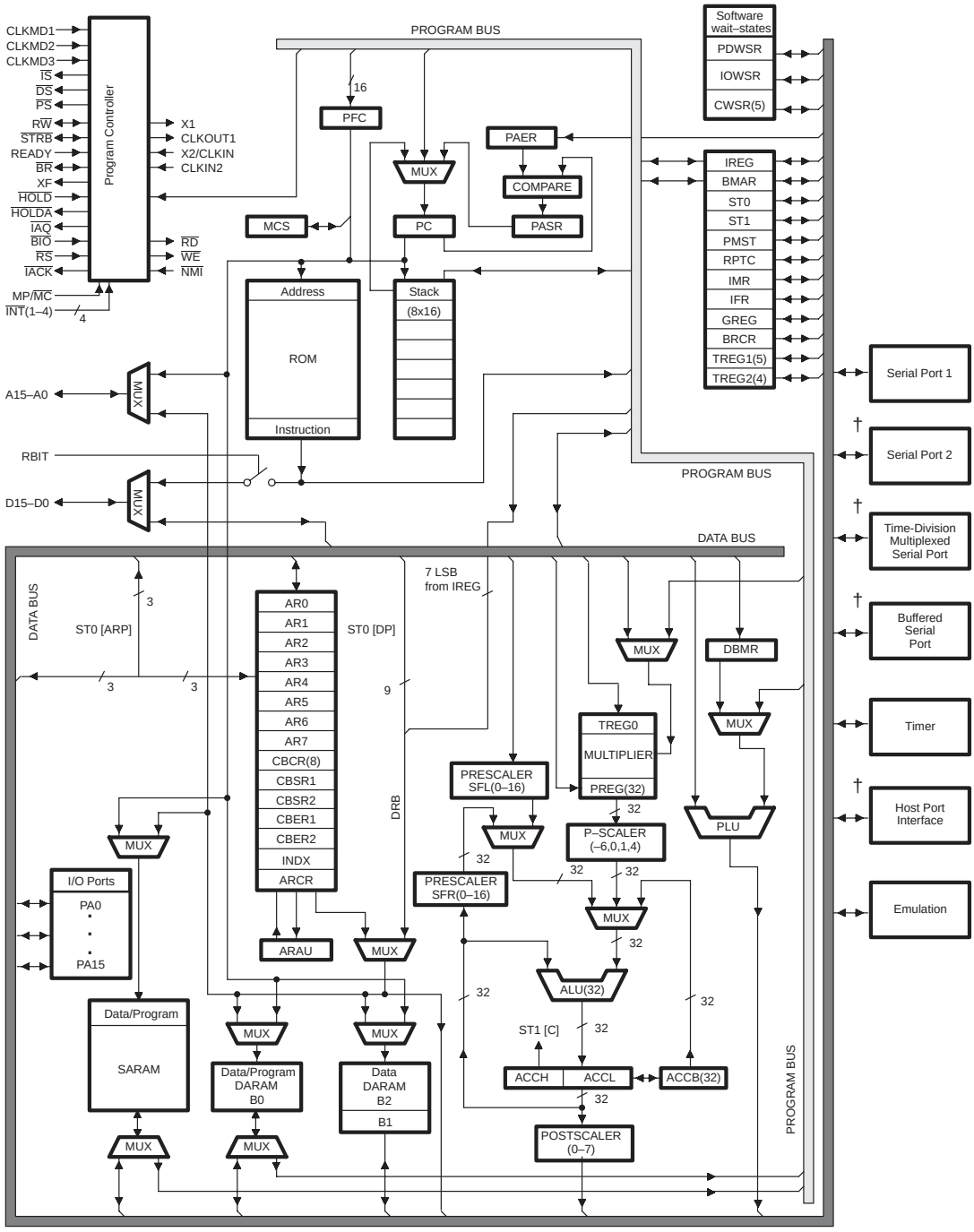
3.1 Functional Overview

The block diagram shown in Figure 3–1 outlines the principal blocks and data paths within the ’C5x. The succeeding sections provide further details of the functional blocks of the CPU.

The internal hardware of the ’C5x executes functions that other processors typically implement in software or microcode. For example, the ’C5x contains hardware for single-cycle 16 $\times$ 16-bit multiplication, data shifting, and address manipulation. This hardware-intensive approach provides computing power previously unavailable on a single chip.

Table 3–1 presents a summary of the ’C5x’s internal hardware. This summary table is alphabetized. The table includes the internal processing elements, registers, and buses. All of the symbols used in the table correspond to the the functional blocks illustrated in Figure 3–1, the succeeding block diagrams in this chapter, and the text throughout this document.

Figure 3–1. Block Diagram of ’C5x DSP – Central Processing Unit (CPU)



Notes: All registers and data lines are 16-bits wide unless otherwise specified.
† Not available on all devices.

Table 3–1. ’C5x CPU Internal Hardware Summary

Symbol	Name
A15–A0	Address bus
ACC(32)	Accumulator
ACCB(32)	Accumulator buffer
ACCH	Accumulator high byte
ACCL	Accumulator low byte
ALU(32)	Arithmetic logic unit
AR0–AR7	Auxiliary registers
ARAU	Auxiliary register arithmetic unit
ARB(3)	Auxiliary register buffer bits
ARCR	Auxiliary register compare register
ARP(3)	Auxiliary register pointer bits
BMAR	Block move address register
BRAF(1)	Block repeat active flag bit
BRCR	Block repeat counter register
C	Carry bit
CALU	Central arithmetic logic unit
CBCR(8)	Circular buffer control register
CBER1, CBER2	Circular buffer end registers
CBSR1, CBSR2	Circular buffer start registers
CNF	Configuration control bit
COMPARE	Compare of program address
D15–D0	Data bus
DATA BUS	Data bus
DBMR	Dynamic bit manipulation register
dma(7)	Data memory address (immediate register)
DP(9)	Data memory page pointer bits

Table 3–1. ’C5x CPU Internal Hardware Summary (Continued)

Symbol	Name
DRB	Direct data memory address bus
GREG	Global memory allocation register
HM(1)	Hold mode bit
IFR	Interrupt flag register
IMR	Interrupt mask register
INDX	Index register
INTM(1)	Interrupt mode bit
IPTR(5)	Interrupt vector pointer bits
IREG	Instruction register
MCS	Microcall stack
MP/$\overline{\text{MC}}$	Microprocessor/microcomputer bit
MULTIPLIER	Multiplier
MUX	Multiplexer
NDX(1)	Enable extra index register bit
OV(1)	Overflow bit
OVLY(1)	RAM overlay bit
OVM(1)	Overflow mode bit
P-SCALER (–6, 0, 1, 4)	Product shifter
PAER	Block repeat program address end register
PASR	Block repeat program address start register
PC	Program counter
PFC	Prefetch counter
PLU	Parallel logic unit
PM(2)	Product shifter mode bits
PMST	Processor mode status register
POSTSCALER(0–7)	Accumulator postscaling shifter

Table 3–1. ’C5x CPU Internal Hardware Summary (Continued)

Symbol	Name
PREG(32)	Product register
PRESCALER, SFL(0–16), SFR(0–16)	Prescaling shifters
PROGRAM BUS	Program bus
RAM(1)	Program RAM enable bit
RPTC	Repeat counter register
ST0, ST1	Status registers
STACK	Stack
SXM(1)	Sign-extension mode bit
TC(1)	Test/control bit
TREG0	Temporary register (multiplicand)
TREG1(5)	Temporary register (dynamic shift count)
TREG2(4)	Temporary register (bit pointer in dynamic bit test)
TRM(1)	Enable multiple temporary registers bit
XF(1)	External flag pin status bit

3.2 Central Arithmetic Logic Unit (CALU)

The CALU components, shown in Figure 3–2, consists of the following:

- 16-bit × 16-bit parallel multiplier
- 32-bit 2s-complement arithmetic logic unit (ALU)
- 32-bit accumulator (ACC)
- 32-bit accumulator buffer (ACCB)
- 0-, 1-, or 4-bit left or 6-bit right shifter
- 0- to 16-bit left barrel shifter
- 0- to 16-bit right barrel shifter
- 0- to 7-bit left barrel shifter

3.2.1 Multiplier, Product Register (PREG), and Temporary Register 0 (TREG0)

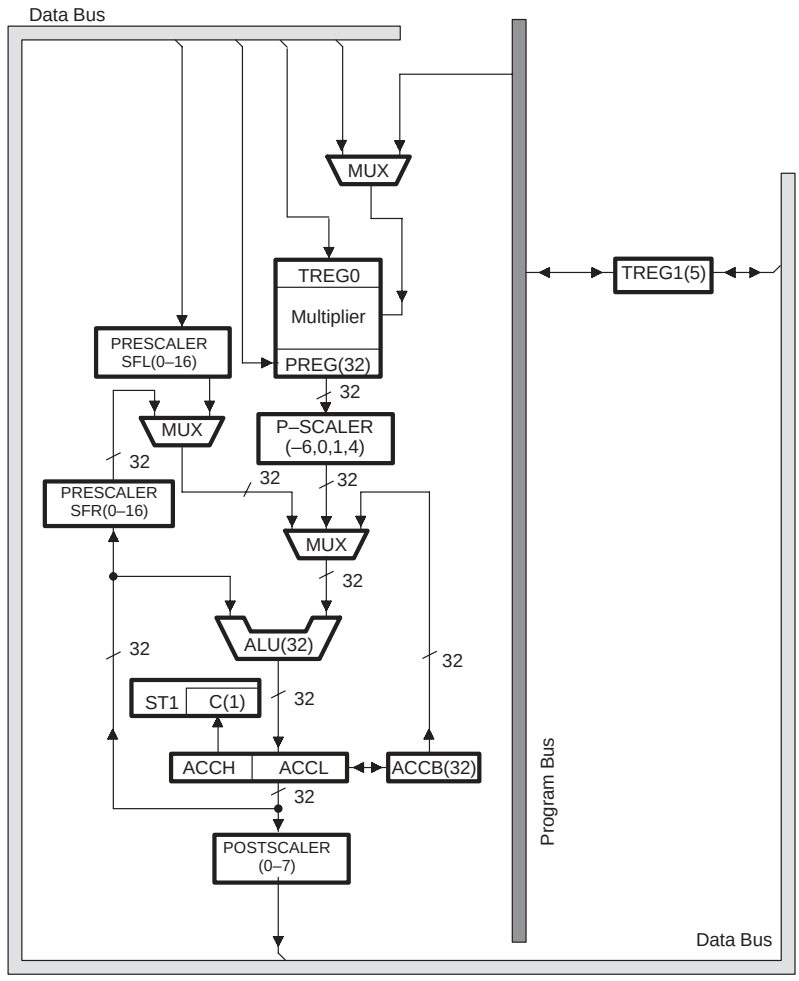
The 16-bit × 16-bit hardware multiplier can compute a signed or an unsigned 32-bit product in a single machine cycle. All multiply instructions except the multiply unsigned (MPYU) instruction perform a signed multiply operation in the multiplier. That is, two numbers being multiplied are treated as 2s-complement numbers, and the result is a 32-bit 2s-complement number.

One input to the multiplier is from memory-mapped temporary register 0 (TREG0), and the other input is from the data bus or the program bus. The 32-bit result from the multiplier is stored in the PREG and is available to the ALU. The ALU uses the 16-bit words taken from data memory or derived from an immediate instruction, or the ALU uses the 32-bit result stored in the PREG to perform arithmetic operations. The ALU can also perform Boolean operations. The 32-bit result from the ALU is stored in the ACC; the ACC also supplies the second input to the ALU. Instructions are provided for storing the high- and low-order accumulator words in memory. The shifters (p-scaler, prescaler, and postscaler) make it possible for the CALU to perform numerical scaling, bit extraction, extended-precision arithmetic, and overflow prevention. These shifters are connected to the output of the PREG and the ACC.

The four product shift modes (PM) at the PREG output are useful for performing multiply/accumulate operations and fractional arithmetic and for justifying fractional products. The PM field of status register ST1 specifies the PM shift mode of the p-scaler:

- If PM = 00_2, the PREG 32-bit output is not shifted when transferred into the ALU or stored.
- If PM = 01_2, the PREG output is left-shifted 1 bit when transferred into the ALU or stored, and the LSB is zero filled. This shift mode compensates for the extra sign bit gained when multiplying two 16-bit 2s-complement numbers.

Figure 3–2. Central Arithmetic Logic Unit



Notes: All registers and data lines are 16-bits wide unless otherwise specified.

- If PM = 10_2, the PREG output is left-shifted 4 bits when transferred into the ALU or stored, and the 4 LSBs are zero filled. This shift mode is used in conjunction with the MPY instruction with a short immediate value (13 bits or less) to eliminate the four extra sign bits gained when multiplying a16-bit number times a 13-bit number.

- If PM = 11_2, the PREG output is right-shifted 6 bits, sign extended, when transferred into the ALU or stored, and the 6 LSBs are lost. This shift mode enables the execution of up to 128 consecutive multiply/accumulates without the possibility of overflow. Note that the product is always sign extended, regardless of the value of the sign extension mode (SXM) bit in ST1.

The PM shifts also occur when the PREG contents are stored to data memory. The PREG contents remain unchanged during the shifts.

The LT (load TREG0) instruction loads TREG0, from the data bus, with the first operand; the MPY instruction provides the second operand for multiplication operations. To perfrom a multiplication with a short or long immediate operand, use the MPY instruction with an immediate operand. A product can be obtained every two cycles except when a long immediate operand is used.

Four multiply/accumulate instructions (MAC, MACD, MADD, and MADS) fully utilize the computational bandwidth of the multiplier, which allows both operands to be processed simultaneously. The data for these operations can be transferred to the multiplier each cycle via the program and data buses. When any of the four multiply/accumulate instructions are used with the RPT or RPTZ instruction, the instruction becomes a single-cycle multiply/accumulate function. In these repeated instructions, the coefficient addresses are generated by the PC while the data addresses are generated by the ARAU. This allows the RPT instruction to sequentially access the values from the coefficient table and step through the data in any of the indirect addressing modes. The RPTZ instruction also clears the ACC and the PREG to initialize the multiply/accumulate operation.

For example, consider multiplying the row of one matrix times the column of a second matrix: there are 10 $\times$ 10 matrices, MTRX1 points to the beginning of the first matrix, INDX = 10, and the current AR points to the beginning of the second matrix:

```
RPTZ   #9           ;For i = 0, i < 10, i++
MAC    MTRX1,*0+    ;PREG=DATA(MTRX1+i) x DATA[MTRX2 +
                    ;(i x INDX)]
                    ;ACC += PREG.
APAC                ;ACC += PREG.
```

The MAC and MACD instructions obtain their coefficient pointer from a long immediate address and are, therefore, 2-word instructions. The MADS and MADD instructions obtain their coefficient pointer from the BMAR and are, therefore, 1-word instructions. When you use the BMAR as a source to the coefficient table, one block of code can support multiple applications, and you can change the long immediate address without modifying executable code. The MACD and MADD instructions include a data move (DMOV) operation that, in conjunction with the fetch of the data multiplicand, writes the data value to the next higher data address.

The MACD and MADD instructions, when repeated, support filter constructs (weighted running averages) so that as the sum-of-products operation is executed, the sample data is shifted in memory to make room for the next sample and to throw away the oldest sample. Circular addressing with MAC and MADS instructions can also be used to support filter implementation.

In the next example, the current AR points to the oldest of the samples; BMAR points to the coefficient table. In addition to initiating the repeat operation, the RPTZ instruction also clears the ACC and the PREG. In this example, the PC is stored in a temporary register while the repeated operation is executed. Next, the PC is loaded with the value stored in BMAR. The program bus is used to address the coefficients and, as the MADD instruction is repeatedly executed, the PC increments to step through the coefficient table. The ARAU generates the address of the sample data.

Indirect addressing with decrement steps through the sample data, starting with the oldest data. As the data is fetched, it is also written to the next higher location in data memory. This operation aligns the data for the next execution of the filter by moving the oldest sample out past the end of the sample's array and making room for the new sample at the beginning of the sample array. The previous product of the PREG is added to the ACC, while the two fetched values are multiplied and the new product value is loaded into the PREG. Note that the DMOV portion of the MACD and MADD instructions does not function with external data memory addresses.

```
RPTZ   #9     ;ACC = PREG = 0. For I = 9 TO 0 Do
MADD   *-     ;SUM A(I) x X(I). X(I+1) = X(I).
APAC          ;FINAL SUM.
```

The MPYU instruction performs an unsigned multiplication that facilitates extended-precision arithmetic operations. The unsigned contents of TREG0 are multiplied by the unsigned contents of the addressed data memory location; the result is placed in PREG. This allows operands larger than 16 bits to be broken down into 16-bit words and processed separately to generate products larger than 32 bits. The square/add (SQRA) and square/subtract (SQRS) instructions pass the same value to both inputs of the multiplier for squaring a data memory value.

After the multiplication of two 16-bit numbers, this 32-bit product is loaded into PREG. The product from the PREG can be transferred to the ALU or to data memory via the store product high (SPH) and store product low (SPL) instructions.

3.2.2 Arithmetic Logic Unit (ALU) and Accumulators

The 32-bit general-purpose ALU and ACC implement a wide range of arithmetic and logical functions, the majority of which execute in a single clock cycle. Once an operation is performed in the ALU, the result is transferred to the ACC, where additional operations, such as shifting, can occur. Data that is input to the ALU can be scaled by the prescaler.

The following steps occur in the implementation of a typical ALU instruction:

1) Data is fetched from memory on the data bus,

2) Data is passed through the prescaler and the ALU, where the arithmetic is performed, and

3) The result is moved into the ACC.

The ALU operates on 16-bit words taken from data memory or derived from immediate instructions. In addition to the usual arithmetic instructions, the ALU can perform Boolean operations, thereby facilitating the bit manipulation ability required of a high-speed controller. One input to the ALU is always supplied by the ACC. The other input can be transferred from the PREG of the multiplier, the ACCB, or the output of the prescaler (that has been read from data memory or from the ACC). After the ALU has performed the arithmetic or logical operation, the result is stored in the ACC. For the following example, assume that ACC = 0, PREG = 0022 2200h, PM = 00_2, and ACCB = 0033 3300h:

```
LACC  #01111h,8 ;ACC = 00111100h. Load ACC from prescaling
                ;shifter
APAC            ;ACC = 00333300h. Add to ACC the
                ;product register.
ADDB            ;ACC = 00666600h. Add to ACC the
                ;accumulator buffer.
```

The 32-bit ACC can be split into two 16-bit segments (ACCH and ACCL) for storage in data memory (see Figure 3–2). A postscaler at the output of the ACC provides a left shift of 0 to 7 places. This shift is performed while the data is being transferred to the data bus for storage. The contents of the ACC remain unchanged. When the postscaler is used on the high word of the ACC (bits 16 – 31), the MSBs are lost and the LSBs are filled with bits shifted in from the low word (bits 0 – 15). When the postscaler is used on the low word, the LSBs are zero filled. For the following example, assume that ACC = FF23 4567h:

```
SACL  TEMP1,7   ;TEMP1 = B380h   ACC = FF234567h.
SACH  TEMP2,7   ;TEMP2 = 91A2h   ACC = FF234567h.
```

The ’C5x supports floating-point operations for applications requiring a large dynamic range. By performing left shifts, the NORM (normalization) instruction normalizes fixed-point numbers contained in the ACC. The four bits of the TREG1 define a variable shift through the prescaler for the add to/load to/subtract from accumulator with shift specified by TREG1 (ADDT/LACT/SUBT) instructions. These instructions denormalize a number (convert it from floating-point to fixed-point) and also execute an automatic gain control (AGC) going into a filter.

The single-cycle 1-bit to 16-bit right shift of the ACC can efficiently align its contents. This shift, coupled with the 32-bit temporary buffer on the ACC, enhances the effectiveness of the CALU in extended-precision arithmetic. The ACCB provides a temporary storage place for a fast save of the ACC. The ACCB can also be used as an input to the ALU. The minimum or maximum value in a string of numbers can be found by comparing the contents of the ACCB with the contents of the ACC. The minimum or maximum value is placed in both registers, and, if the condition is met, the carry bit (C) is set. The minimum and maximum functions are executed by the CRLT and CRGT instructions, respectively. These operations are signed arithmetic operations. In the next example, assume that ACC = 1234 5678h and ACCB = 7654 3210h:

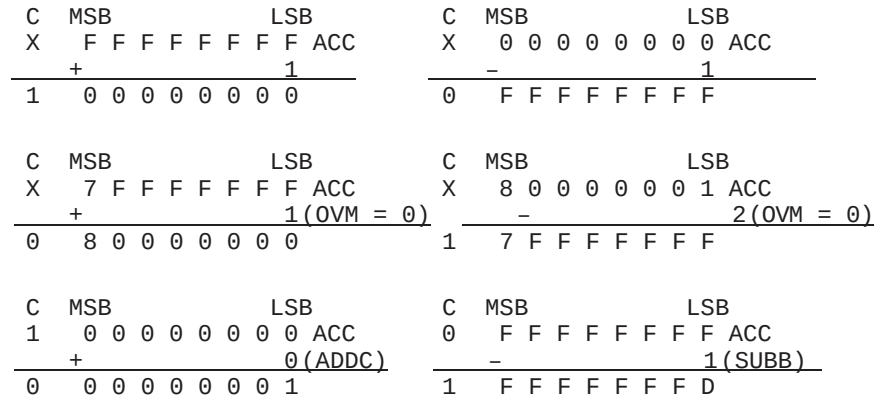
```
CRLT   ;ACC = ACCB = 12345678h. C = 1.
CRGT   ;ACC = ACCB = 76543210h. C = 0.
```

The ACC overflow saturation mode can be enabled by setting and disabled by clearing the overflow mode (OVM) bit of ST0. When the ACC is in the overflow saturation mode and an overflow occurs, the overflow flag is set and the ACC is loaded with either the most positive or the most negative value representable in the ACC, depending upon the direction of the overflow. The value of the ACC upon saturation is 7FFF FFFFh (positive) or 8000 0000h (negative). If the OVM bit is cleared and an overflow occurs, the overflowed results are loaded into the ACC without modification. Note that logical operations cannot result in overflow.

The ’C5x can execute a variety of branch instructions that depend on the status of the ALU and the ACC. For example, execution of the instruction BCND can depend on a variety of conditions in the ALU and the ACC. The BACC instruction allows branching to an address stored in the ACC. The bit test instructions (BITT and BIT) facilitate branching on the condition of a specified bit in data memory.

The ACC has an associated carry bit that is set or cleared, depending on various operations within the ’C5x. The carry bit allows more efficient computation of extended-precision products and additions or subtractions; it is also useful in overflow management. The carry bit is affected by most arithmetic instructions as well as the single-bit shift and rotate instructions. The carry bit is not affected by loading the ACC, logical operations, or other nonarithmetic or control instructions. Examples of carry bit operations are shown in Figure 3–3.

Figure 3–3. Examples of Carry Bit Operations



The value added to or subtracted from the ACC can come from the prescaler, ACCB, or PREG. The carry bit is set if the result of an addition or accumulation process generates a carry; it is cleared if the result of a subtraction generates a borrow. Otherwise, it is cleared after an addition or set after a subtraction.

The add to ACC with carry (ADDC) and add ACCB to ACC with carry (ADCB) instructions use the previous value of carry in their addition operation. The subtract from ACC with borrow (SUBB) and subtract ACCB from ACC with borrow (SBBB) instructions use the logical inversion of the previous value of carry.

The one exception to the operation of the carry bit is in the use of ADD with a shift count of 16 (add to ACCH) and SUB with a shift count of 16 (subtract from ACCH). These instructions can generate a carry or a borrow, but they will not clear a carry or borrow, as is normally the case if a carry or borrow is not generated. This feature is useful for extended-precision arithmetic.

Two conditional operands, C and NC, are provided for branching, calling, returning, and conditionally executing according to the status of the carry bit. The CLRC, LST #1, and SETC instructions can be used to load the carry bit. The carry bit is set on a reset.

The 1-bit shift to the left (SFL) or right (SFR) and the rotate to the left (ROL) or right (ROR) instructions shift or rotate the contents of the ACC through the

carry bit. The SXM bit affects the definition of the shift accumulator right (SFR) instruction. When SXM = 1, SFR performs an arithmetic right shift, maintaining the sign of the ACC data. When SXM = 0, SFR performs a logical shift, shifting out the LSBs and shifting in a 0 for the MSB. The shift accumulator left (SFL) instruction is not affected by the SXM bit and behaves the same in both cases, shifting out the MSB and shifting in a 0. The RPT and RPTZ instructions can be used with the shift and rotate instructions for multiple-bit shifts.

The SFLB, SFRB, RORB, and ROLB instructions can shift or rotate the 65-bit combination of the ACC, ACCB, and carry bit as described above.

The ACC can also be shifted 0–31 bits right in two instruction cycles or 1–16 bits right in one cycle. The bits shifted out are lost, and the bits shifted in are either 0s or copies of the original sign bit, depending on the value of the SXM bit. A shift count of 1 to 16 is embedded in the instruction word of the BSAR instruction. For example, let ACC = 1234 5678h:

```
BSAR   7              ;ACC = 02468ACEh.
```

The right shift can also be controlled via TREG1. The SATL instruction shifts the ACC by 0–15 bits, as defined by bits 0–3 of TREG1. The SATH instruction shifts the ACC 16 bits to the right if bit 4 of TREG1 is a 1. The following code sequence executes a 0- to 31-bit right shift of the ACC, depending on the shift count stored at SHIFT. For example, consider the value stored at SHIFT = 01Bh and ACC = 1234 5678h:

```
LMMR   TREG1,SHIFT  ;TREG1 = shift count 0 – 31. TREG1 = 1B
SATH                ;If shift count > 15, then ACC >> 16
                    ;ACC = 00001234
SATL                ;ACC >> shift count. ACC = 00000002
```

3.2.3 Scaling Shifters and Temporary Register 1 (TREG1)

The prescaler has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU (see Figure 3–2). The prescaler produces a left shift of 0 to 16 bits on the input data. The shift count is specified by a constant embedded in the instruction word or by the value in TREG1. The LSBs of the output are filled with 0s; the MSBs can be filled with 0s or sign-extended, depending upon the value of the SXM bit of ST1.

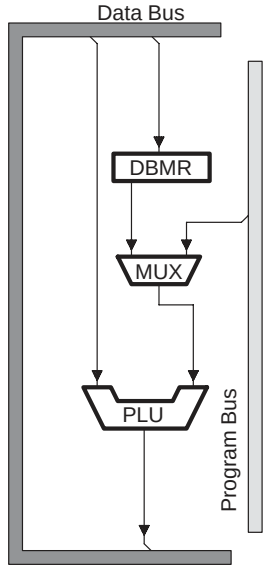
The p-scaler and postscaler make it possible for the CALU to perform numerical scaling, bit extraction, extended-precision arithmetic, and overflow prevention. These shifters are connected to the output of the PREG and the ACC (see Figure 3–2 on page 3-8).

3.3 Parallel Logic Unit (PLU)

The parallel logic unit (PLU) can directly set, clear, test, or toggle multiple bits in a control/status register or any data memory location. The PLU provides a direct logic operation path to data memory values without affecting the contents of the ACC or the PREG (see Figure 3–4).

The PLU executes a read-modify-write operation on data stored in data space. First, one operand is fetched from data memory space, and the second is fetched from a long immediate on the program bus or from the dynamic bit manipulation register (DBMR). Then, the PLU executes a logical operation on the two operands as defined by the instruction. The result is written to the same data memory location from which the first operand was fetched.

Figure 3–4. Parallel Logic Unit Block Diagram



Note: All registers and data lines are 16-bits wide unless otherwise specified.

The PLU makes it possible to directly manipulate bits in any location in data memory space by ANDing, ORing, exclusive-ORing, or loading a 16-bit long immediate value to a data location. For example, to use AR1 for circular buffer 1 and AR2 for circular buffer 2 but not enable the circular buffers, initialize the circular buffer control register (CBCR) by executing the following code:

```
SPLK #021h,CBCR          ;Store peripheral long immediate
                         ;(DP = 0).
```

Next, enable circular buffers 1 and 2 by executing the code:

```
OPL  #088h,CBCR          ;Set bit 7 and bit 3 in CBCR.
```

To test for individual bits in a specific register or data word, use the BIT instruction; however, to test for a pattern of bits, use the compare parallel long immediate (CPL) instruction. If the data value is equal to the long immediate value, then the test/control (TC) bit in ST1 is set. The TC bit is set if the result of any PLU instruction is 0.

The set, clear, and toggle functions can be executed with a 16-bit dynamic register value instead of the long immediate value. This is done with the following three instructions: AND DBMR register to data (APL), OR DBMR register to data (OPL), and exclusive-OR DBMR register to data (XPL).

The TC bit is also set by the APL, OPL, and XPL instructions if the result of the PLU operation (value written back into data memory) is 0. This allows bits to be tested and cleared simultaneously. For example,

```
APL    #0FF00h,TEMP        ;Clear low byte and check for
                           ;bits set in high byte.
BCND   HIGH_BITS_SET,NTC   ;If bits active in high byte,
                           ;then branch.
```

or

```
XPL    #1,TEMP             ;Toggle bit 0.
BCND   BIT_SET,TC          ;If bit was set, branch. If not,
                           ;bit set now.
```

In the first example, the low byte of a flag word is cleared while the high byte is checked for any active flags (bits = 1). If none of the flags in the high byte is set, then the resulting APL operation yields a 0 to TEMP and the TC bit is set. If any of the flags in the high byte are set, then the resulting APL operation yields a nonzero value to TEMP and the TC bit is cleared. Therefore, the conditional branch (BCND) following the APL instruction branches if any of the bits in the high byte are nonzero. The second example tests the flag. If the flag is low, the flag is set high; if the flag is high, the flag is cleared and the branch is taken. The PLU instructions can operate anywhere in data address space, so they can operate with flags stored in RAM locations as well as in control registers for both on- and off-chip peripherals. The PLU instructions are listed in Table 6–6 on page 6-14.

3.4 Auxiliary Register Arithmetic Unit (ARAU)

The auxiliary register file contains eight memory-mapped auxiliary registers (AR0–AR7), which can be used for indirect addressing of the data memory or for temporary data storage. Indirect auxiliary register addressing (see Figure 3–5) allows placement of the data memory address of an instruction operand into one of the AR. The ARs are pointed to by a 3-bit auxiliary register pointer (ARP) that is loaded with a value from 0–7, designating AR0–AR7, respectively. The ARs and the ARP can be loaded from data memory, the ACC or the PREG or by an immediate operand defined in the instruction. The contents of the ARs can be stored in data memory or used as inputs to the CALU. The memory-mapped ARs reside in data page 0, as described in subsection 8.3.2, *Local Data Memory Address Map*, on page 8-17.

The auxiliary register file (AR0–AR7) is connected to the auxiliary register arithmetic unit (ARAU), shown in Figure 3–6. The ARAU can autoindex the current AR while the data memory location is being addressed; it indexes either by ±1 or by the contents of the index register (INDX). As a result, the CALU is not needed for address manipulation when tables of information are accessed; it is free for other operations in parallel. For more advanced address manipulation, such as multidimensional array addressing, the CALU can directly read from or write to the ARs.

Figure 3–5. Indirect Auxiliary Register Addressing Example

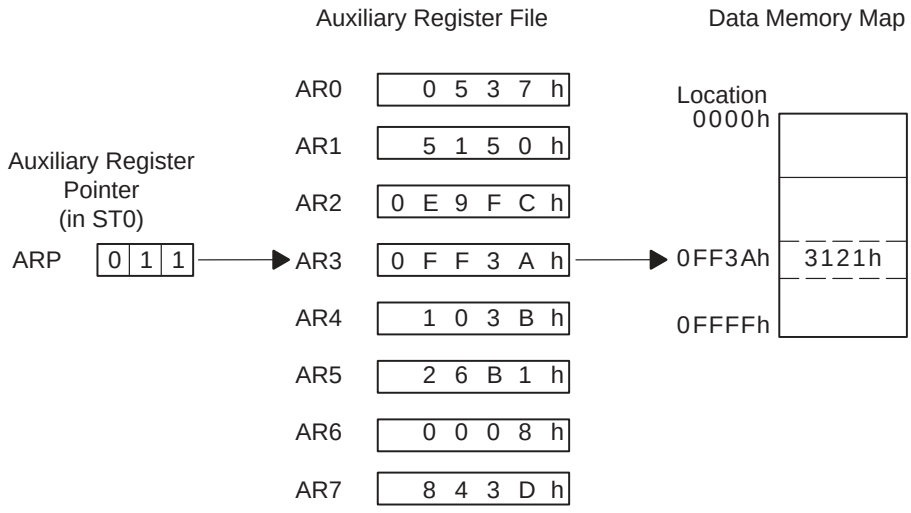
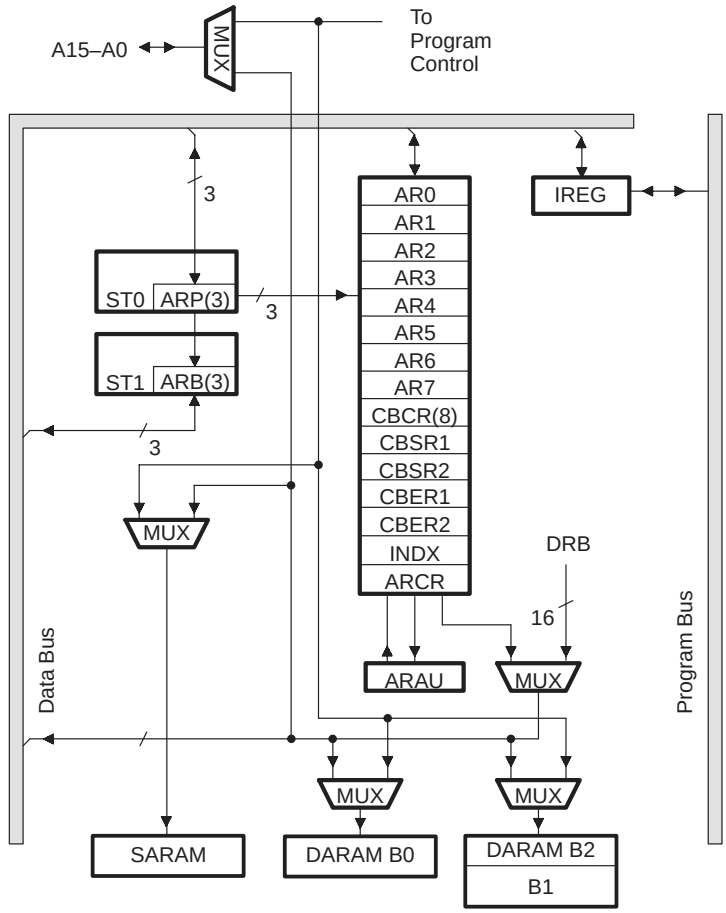


Figure 3–6. Auxiliary Register Arithmetic Unit



Notes: All registers and data lines are 16-bits wide unless otherwise specified.

The ARAU updates the ARs during the decode phase (second stage) of the pipeline, while the CALU writes during the execution phase (fourth stage). Therefore, the two instructions that immediately follow the CALU write to an AR should not use the same AR for address generation. See Chapter 7, *Pipeline*, for more details.

As shown in Figure 3–6, the INDX, auxiliary register compare register (ARCR), or eight LSBs of the instruction register (IREG) can be used as one of the inputs to the ARAU. The other input is provided by the contents of the current AR pointed to by ARP. Table 3–2 defines the functions of the ARAU.

Table 3–2. Auxiliary Register Arithmetic Unit Functions

Function	Description
Current AR + INDX → Current AR	Index the current AR by adding an unsigned 16-bit integer contained in INDX. Example: ADD *0+
Current AR – INDX → Current AR	Index the current AR by subtracting an unsigned 16-bit integer contained in INDX. Example: ADD *0–
Current AR + 1 → Current AR	Increment the current AR by 1. Example: ADD *+
Current AR – 1 → Current AR	Decrement the current AR by 1. Example: ADD *–
Current AR → Current AR	Do not modify the current AR. Example: ADD *
Current AR + IR(7–0) → Current AR	Add an 8-bit immediate value to current AR. Example: ADRK #55h
Current AR – IR(7–0) → Current AR	Subtract an 8-bit immediate value from the current AR. Example: SBRK #55h
Current AR + rc(INDX) → Current AR	Bit-reversed indexing; add INDX with reversed-carry (rc) propagation. Example: ADD *BR0+
Current AR – rc(INDX) → Current AR	Bit-reversed indexing; subtract INDX with reversed-carry (rc) propagation. Example: ADD *BR0–

If (Current AR) = (ARCR), then TC = 1
If (Current AR) < (ARCR), then TC = 1
If (Current AR) > (ARCR), then TC = 1
If (Current AR) ≠ (ARCR), then TC = 1
	Compare the current AR to ARCR and, if the condition is true, then set the TC bit of the status register ST1. If false, then clear the TC bit. Example: CMPR 3

If (Current AR) = (CBER), then Current AR = CBSR	If the current AR is at the end of circular buffer, reload the start address. The test for this condition is performed before the execution of the AR modification. Example: ADD *+

The INDX can be added to or subtracted from the current AR on any AR update cycle. The INDX can be used to increment or decrement the address in steps larger than 1; this is useful for operations such as addressing down a matrix column. The ARCR limits blocks of data and supports logical comparisons between the current AR and ARCR in conjunction with the CMPR instruction. Note that the ’C2x uses AR0 for this implementation. After reset, you can use the load auxiliary register (LAR) instruction to load AR0; if the enable extra index register (NDX) bit in the PMST is set, LAR also loads INDX and ARCR to maintain compatibility with the ’C2x.

Because the ARs are memory-mapped, the CALU can act directly upon them and use more advanced indirect addressing techniques. For example, the multiplier can calculate the addresses of 3-dimensional matrices. After a CALU load of the AR, there is, however, a 2-instruction-cycle delay before the ARs can be used for address generation. The INDX and ARCR are accessible via the CALU, regardless of the condition of the NDX bit (that is, SAMM ARCR writes only to the ARCR).

The ARAU can serve as an additional general-purpose arithmetic unit because the auxiliary register file can directly communicate with data memory. The ARAU implements 16-bit unsigned arithmetic, whereas the CALU implements 32-bit 2s-complement arithmetic. The BANZ and BANZD instructions permit the ARs to be used as loop counters.

The 3-bit auxiliary register pointer buffer (ARB), shown in Figure 3–6, stores the ARP on subroutine calls when the automatic context switch feature of the ’C5x is not used.

Two circular buffers can operate at a given time and are controlled via the circular buffer control register (CBCR). Upon reset (rising edge of $\overline{RS}$), both circular buffers are disabled. To define a circular buffer, load CBSR1 or CBSR2 with the start address of the buffer and CBER1 or CBER2 with the end address; then load the AR to be used with the circular buffer with an address between the start and end addresses. Finally, load CBCR with the appropriate AR number and set the enable (CENB1 or CENB2) bit.

Do not use the same AR to access both circular buffers or unexpected results will occur.

As the address is stepping through the circular buffer, the AR value is compared against the value contained in CBER prior to the update to the AR value. If the current AR value and the CBER are equal and an AR modification occurs, the value contained in CBSR is automatically loaded into the AR. If the values in the CBER and the AR are not equal, the AR is modified as specified.

Circular buffers can be used with either increment- or decrement-type updates. If increment is used, then the value in CBER must be larger than the value in CBSR. If decrement is used, the value in CBER must be smaller than the value in CBSR. The other indirect addressing modes can be used; however, the ARAU tests only for the condition current AR = CBER. The ARAU does not detect an AR update that steps over the value contained in CBER. See Section 5.6, *Circular Addressing*, on page 5-21 for more details.

3.5 Summary of Registers

CPU registers (except ST0 and ST1), peripheral registers, and I/O ports occupy data memory space.

3.5.1 Auxiliary Registers (AR0–AR7)

The eight 16-bit auxiliary registers (AR0–AR7) can be accessed by the CALU and modified by the ARAU or the PLU. The primary function of the ARs is to provide a 16-bit address for indirect addressing to data space. However, the ARs can also be used as general-purpose registers or counters. Section 5.2, *Indirect Addressing*, on page 5-4 describes how the ARs are used in indirect addressing. Use of ARs is described in Section 3.4 on page 3-17.

3.5.2 Auxiliary Register Compare Register (ARCR)

The 16-bit ARCR is used for address boundary comparison. The CMPR instruction compares the ARCR to the selected AR and places the result of the compare in the TC bit of ST1. Section 5.2, *Indirect Addressing*, on page 5-4 describes how the ARCR can be used in memory management. See also Section 3.4 on page 3-17.

3.5.3 Block Move Address Register (BMAR)

The 16-bit BMAR holds an address value to be used with block moves and multiply/accumulate operations. This register provides the 16-bit address for an indirect-addressed second operand. See Section 5.4, *Dedicated-Register Addressing*, on page 5-17.

3.5.4 Block Repeat Registers (RPTC, BRCR, PASR, PAER)

The 16-bit repeat counter register (RPTC) holds the repeat count in a repeat single-instruction operation and is loaded by the RPT and RPTZ instructions. See Section 4.6, *Single Instruction Repeat Function*, on page 4-22.

Although the RPTC is a memory-mapped register, you should avoid writing to this register. Writing to this register can cause undesired results.

The 16-bit block repeat counter register (BRCR) holds the count value for the block repeat feature. This value is loaded before a block repeat operation is initiated. The value can be changed while a block repeat is in progress; however, take care to avoid infinite loops. The block repeat program address start register (PASR) indicates the 16-bit address where the repeated block of code starts. The block repeat program address end register (PAER) indicates the 16-bit address where the repeated block of code ends. The PASR and PAER are loaded by the RPTB instruction. Block repeats are described in Section 4.7, *Block Repeat Function*, on page 4-31.

3.5.5 Buffered Serial Port Registers (ARR, AXR, BKR, BKX, SPCE)

The buffered serial port (BSP) is available on ’C56 and ’C57 devices. The BSP comprises a full-duplex, double-buffered serial port interface and an autobuffering unit (ABU). The BSP has a 2K-word buffer, which resides in the ’C5x internal memory. Five registers control and operate the BSP. The 16-bit BSP control extension register (SPCE) contains the mode control and status bits of the BSP. The 11-bit BSP address receive register (ARR) and 11-bit BSP receive buffer size register (BKR) support address generation for writing to the data receive register (DRR) in the ’C5x internal memory. The 11-bit BSP address transmit register (AXR) and 11-bit BSP transmit buffer size register (BKX) support address generation for reading a word from the ’C5x internal memory to the data transmit register (DXR). The BSP is described in Section 9.8, *Buffered Serial port (BSP) Interface*, on page 9-53.

3.5.6 Circular Buffer Registers (CBSR1, CBER1, CBSR2, CBER2, CBCR)

The ’C5x devices support two concurrent circular buffers operating in conjunction with user-specified auxiliary registers. Two 16-bit circular buffer start registers (CBSR1 and CBSR2) indicate the address where the circular buffer starts. Two 16-bit circular buffer end registers (CBER1 and CBER2) indicate the address where the circular buffer ends. The 16-bit circular buffer control register (CBCR) controls the operation of these circular buffers and identifies the auxiliary registers to be used. Section 5.6, *Circular Addressing*, on page 5-21 describes how circular buffers can be used in memory management. Section 3.4 on page 3-17 describes how circular buffer registers are used in addressing. See also subsection 4.4.1, *Circular Buffer Control Register (CBCR)*, on page 4-6.

3.5.7 Dynamic Bit Manipulation Register (DBMR)

The 16-bit DBMR is used in conjunction with the PLU as a dynamic (execution-time programmable) mask register. The DBMR is described in Section 3.3 on page 3-15.

3.5.8 Global Memory Allocation Register (GREG)

The 16-bit GREG allocates parts of the local data space as global memory and defines what amount of the local data space will be overlayed by global data space. See Section 8.4, *Global Data Memory*, on page 8-20.

3.5.9 Host Port Interface Registers (HPIC, HPIA)

The 8-bit wide parallel host port interface (HPI) is available on the 'C57 device. The HPI interfaces a host processor to the 'C57 device. The HPI control register (HPIC) holds the control word. The host processor addresses HPI memory via the HPI address register (HPIA). See Section 9.10, *Host Port Interface ('C57S and 'LC57 only)*, on page 9-87.

3.5.10 Index Register (INDX)

The 16-bit INDX is used by the ARAU as a step value (addition or subtraction by more than 1) to modify the address in the ARs during indirect addressing. For example, when the ARAU steps across a row of a matrix, the indirect address is incremented by 1. However, when the ARAU steps down a column, the address is incremented by the dimension of the matrix. The ARAU can add or subtract the value stored in the INDX from the current AR as part of the indirect address operation. INDX can also map the dimension of the address block used for bit-reversal addressing. Section 5.2, *Indirect Addressing*, on page 5-4 describes how the INDX can be used in memory management. See also Section 3.4 on page 3-17.

3.5.11 I/O Space (PA0–PA15)

The I/O space makes it possible to address 16 locations (50h–5Fh) of I/O space via the addressing modes of the local data space. This means that these locations can be read directly into the CALU or written from the ACC. It also means that these locations can be acted upon by the PLU or addressed via the memory-mapped addressing mode. The locations can also be addressed with the IN and OUT instructions.

3.5.12 Instruction Register (IREG)

The 16-bit IREG holds the opcode of the instruction being executed. The IREG is used during program control.

3.5.13 Interrupt Registers (IMR, IFR)

The 16-bit interrupt mask register (IMR) individually masks specific interrupts at required times. The 16-bit interrupt flag register (IFR) indicates the current status of the interrupts. The status of the interrupts is updated regardless of the IMR and INTM bit in the ST0. Interrupts are described in Section 4.8, *Interrupts*, on page 4-36.

3.5.14 Processor Mode Status Register (PMST)

The 16-bit PMST contains status and control information for the ’C5x device. Subsection 8.2.1, *Program Memory Configurability*, on page 8-7 and subsection 8.3.1, *Local Data Memory Configurability*, on page 8-15 describe how the PMST configures memory. See also subsection 4.4.2, *Processor Mode Status Register (PMST)*, on page 4-7.

3.5.15 Product Register (PREG)

The 32-bit PREG holds the result of a multiply operation. The high and low words of PREG can be accessed individually. See subsection 3.2.1 on page 3-7.

3.5.16 Serial Port Interface Registers (SPC, DRR, DXR, XSR, RSR)

Five registers control and operate the serial port interface. The 16-bit serial port control register (SPC) contains the mode control and status bits of the serial port. The 16-bit data receive register (DRR) holds the incoming serial data, and the 16-bit data transmit register (DXR) holds the outgoing serial data. The 16-bit data transmit shift register (XSR) controls the shifting of the data from the DXR to the output pin. The 16-bit data receive shift register (RSR) controls the storing of the data from the input pin to the DRR. The serial port is described in Section 9.7, *Serial Port Interface*, on page 9-23.

3.5.17 Software-Programmable Wait-State Registers (PDWSR, IOWSR, CWSR)

The software wait states are determined by three registers. These registers serve different purposes on different devices. On most ’C5x devices the 16-bit program/data wait-state register (PDWSR) contains the wait-state count for the eight 16K-word blocks of program and data memory. The PDWSR is divided into eight 2-bit wait-state fields assigned to each 16K-word block. The I/O space is mapped into the 16-bit I/O wait-state register (IOWSR) under control of the 5-bit wait-state control register (CWSR). The CWSR determines the range of wait states selected. The BIG bit in the CWSR determines how the I/O space is partitioned. If the BIG bit is cleared, the IOWSR is divided into eight pairs of I/O ports with the 2-bit wait-state fields assigned to each pair of port addresses. If the BIG bit is set, the I/O space is divided into eight 8K-word blocks with each having its own 2-bit wait-state field, similar to PDWSR. For the ’C52, ’LC56, ’C57S, and ’LC57 devices, the program, data, and I/O space wait states are each specified by a single (3-bit) wait-state value. Each memory space can be independently set to 0–7 wait states by a 3-bit wait-state field in PDWSR. See Section 9.4, *Software-Programmable Wait-State Generators*, on page 9-13.

3.5.18 Status Registers (ST0, ST1)

The two 16-bit status registers contain status and control bits for the CPU and are described in subsection 4.4.3, *Status Registers (ST0 and ST1)*, on page 4-10.

3.5.19 Temporary Registers (TREG0, TREG1, TREG2)

The 16-bit TREG0 holds one of the multiplicands of the multiplier. TREG0 can also be loaded via the CALU with the following instructions: LT, LTA, LTD, LTP, LTS, SQRA, SQRS, MAC, MACD, MADS, and MADD. The 5-bit TREG1 holds a dynamic (execution-time programmable) shift count for the prescaling shifter. The 4-bit TREG2 holds a dynamic bit address for the BITT instruction. The TREG0 is described in subsection 3.2.1 on page 3-7.

Software compatibility can be maintained with the ’C2x by clearing the enable multiple TREGs (TRM) bit in the PMST. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x.

3.5.20 Timer Registers (TIM, PRD, TCR)

Three registers control and operate the timer. The timer counter register (TIM) gives the current count of the timer. The timer period register (PRD) defines the period for the timer. The 16-bit timer control register (TCR) controls the operations of the timer. See Section 9.3, *Timer*, on page 9-9.

3.5.21 TDM Serial Port Registers (TRCV, TDXR, TSPC, TCSR, TRTA, TRAD, TRSR)

The time-division-multiplexed (TDM) serial port interface is a feature superset of the serial port interface and supports applications that require serial communication in a multiprocessing environment. Six registers control and operate the TDM serial port interface. The 16-bit TDM serial port control register (TSPC) contains the mode control and status bits of the TDM serial port interface. The 16-bit TDM data receive register (TRCV) holds the incoming TDM serial data, and the 16-bit TDM data transmit register (TDXR) holds the outgoing TDM serial data. The 16-bit TDM data receive shift register (TRSR) controls the storing of the data, from the input pin, to the TRCV. The 16-bit TDM channel select register (TCSR) specifies in which time slot(s) each ’C5x device is to transmit. The 16-bit TDM receive/transmit address register (TRTA) specifies in the eight LSBs (RA0–RA7) the receive address of the ’C5x device and in the eight MSBs (TA0–TA7) the transmit address of the ’C5x device. The 16-bit TDM receive address register (TRAD) contains various information regarding the status of the TDM address line (TADD). See Section 9.9, *Time-Division Multiplexed (TDM) Serial Port Interface*, on page 9-74.

Chapter 4

Program Control

Program control on the TMS320C5x is provided by the program counter, hardware stack, repeat counters, status registers, program counter-related hardware, and several software mechanisms. Software mechanisms used for program control include branches, calls, conditional instructions, repeat instructions, reset, and interrupts.

Topic **Page**

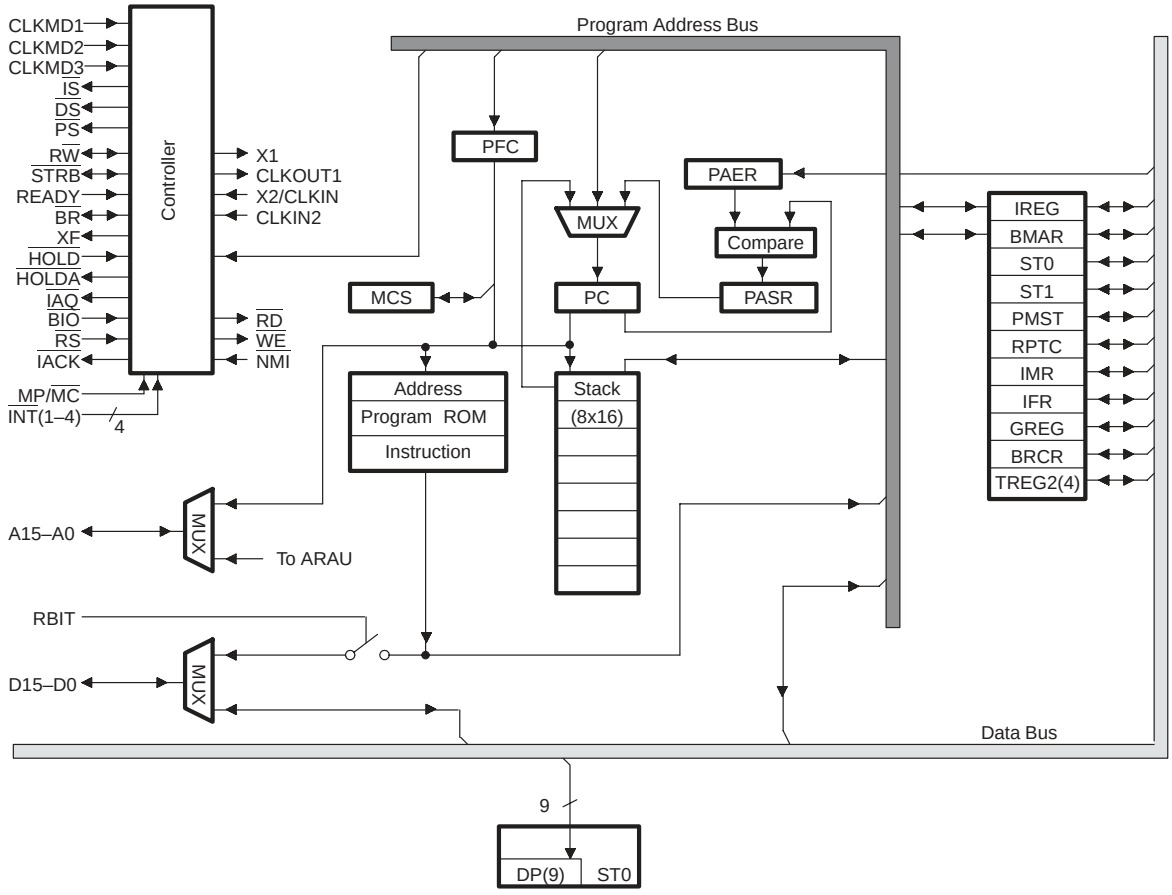
4.1 Program Counter (PC)

The ’C5x has a 16-bit program counter (PC) which contains the address of internal or external program memory used to fetch instructions.

The PC addresses program memory, either on-chip or off-chip, via the program address bus (PAB). Through the PAB, an instruction is loaded into the instruction register (IREG). Then the PC is ready to start the next instruction fetch cycle. Refer to Figure 4–1 for a functional block diagram of the program control elements.

The PC is loaded in a number of ways. Table 4–1 shows what address is loaded into the PC, depending on the code operation performed.

Figure 4–1. Program Control Functional Block Diagram



Notes: All registers and data lines are 16 bits wide unless otherwise specified.

Table 4–1. Address Loading Into the Program Counter

Code Operation	Address Loaded to the PC
Sequential code	The PC is loaded with PC + 1.
Branch (B instruction)	The PC is loaded with the long immediate value directly following the branch instruction.
Subroutine call	The PC + 2 is pushed onto the stack and then the PC is loaded with the long immediate value directly following the call instruction. The return instruction pops the stack back into the PC to return to the calling or interrupting sequence of code.
Software (INTR, TRAP, or NMI instruction) or interrupt trap	The PC is loaded with the address of the appropriate interrupt vector.
Computed GOTO	The content of the accumulator low byte (ACCL) is loaded into the PC. The BACC (branch to location specified by the accumulator) or CALA (call subroutine at location specified by the accumulator) instructions can be used to perform GOTO operations.
BLDD, BLDP, BLPD, MAC, or MACD instruction	The PC is loaded with the a long immediate address.
BACC, BACCD, CALA, TBLR, or TBLW instruction	The PC is loaded with the contents of the accumulator low byte (ACCL).
BLDD, BLDP, BLPD, MADD, or MADS instruction	The PC is loaded with the content of the block move address register (BMAR).
End of a block repeat loop	The PC is loaded with the content of the block repeat program address start register (PASR).
Return instruction	The PC is loaded with the top of the stack.

The PC can also be loaded with coefficients residing in program memory for some instructions used with the repeat operation (see Section 4.6, *Single Instruction Repeat Function,* on page 4-22). In a repeat operation, once the instruction is repeated, it is no longer prefetched, and the PC can be used to address program memory sequentially. The multiply/accumulate instructions (MAC, MACD, MADD, and MADS), memory move from data-to-data instruction (BLDD), memory move from program-to-data instructions (BLPD and TBLR), and memory move from data-to-program instructions (BLDP and TBLW), use this capability.

4.2 Hardware Stack

The stack which is 16 bits wide and 8 levels deep, is accessible via the PUSH and POP instructions. Whenever the contents of the PC are pushed onto the top of the stack (TOS), the previous contents of each level are pushed down, and the bottom (eighth) location of the stack is lost. Therefore, data is lost if more than eight successive pushes occur before a pop. The reverse happens on pop operations. Any pop after seven sequential pops yields the value at the bottom stack level, and then all of the stack levels contain the same value. Two additional instructions — PSHD (push a data memory value onto TOS) and POPD (pop a value from TOS to data memory) — are also available. These instructions allow a stack to be built in data memory for the nesting of subroutines and interrupts beyond eight levels.

The software can use the stack to save and restore context or for other purposes through the following software instructions:

- ❏ POP, which pops a value from the stack to the accumulator low byte
- ❏ POPD, which pops a value from the stack to a data memory address
- ❏ PSHD, which pushes a data-memory value into the stack
- ❏ PUSH, which pushes the contents of the accumulator low byte into the stack

The stack is used during interrupts and subroutines to save and restore the PC contents. When a subroutine is called (CALA, CALAD, CALL, CALLD, CC, or CCD instruction) or an interrupt occurs (hardware interrupt, NMI, INTR, or TRAP instruction), the return address is automatically saved in the stack (a PUSH operation). When a subroutine returns (RET, RETC, RETCD, RETD, RETE, or RETI instruction), the return address is retrieved from the stack (a POP operation) and loaded into the PC.

4.3 Program-Memory Address Generation

The program memory space contains the code for applications and holds table information and immediate operands. The program memory is accessed only by the program address bus (PB). The address for this bus is generated by the program counter (PC) when instructions and long immediate operands are accessed. The PB can also be loaded with a long immediate operand and the lower 16-bit word of the accumulator for block transfers, multiply/accumulates, table reads and writes, branching, and subroutine calls.

The ’C5x fetches instructions by putting the PC on the PAB and reading the appropriate location in memory. While the read is executing, the PC is incremented for the next fetch. If a program address discontinuity (for example, a branch, a call, a return, an interrupt, or a block repeat) occurs, the appropriate address is loaded into the PC. The PC is also loaded when operands are fetched from program memory. Operands are fetched from program memory when the ’C5x reads from (TBLR) or writes to (TBLW) a table or when it transfers data to (BLPD) or from (BLDP) data space. Some instructions (MAC, MACD, MADD, and MADS) use the program bus to fetch a second multiplicand.

The PC can address data stored in either program or data space. This makes it possible, within repeated instructions, to fetch a second operand in parallel with the data bus for 2-operand operations. For repeated instructions, the array is sequentially accessed by the PAB by incrementing the PC. The block transfer instructions (BLDD, BLDP, and BLPD) use both buses so that the pipeline structure can read the next operand while writing to the current one. The BLPD instruction loads the PC with either the long immediate address or with the BMAR contents and then uses the PB to fetch the source data from program space for the block move operation. The BLDP executes in the same way, except that the PAB is used for the destination operation. The BLDD instruction uses the PAB to address data space.

The TBLR and TBLW instructions operate like the BLPD and BLDP instructions, respectively, except that the PC is loaded with the accumulator low byte instead of the long immediate address or the BMAR contents. This allows look-up table operations. The multiply/accumulate operations (MAC, MACD, MADD, and MADS) use the PAB to address their coefficient table. The MAC and MACD instructions load the PC with the long immediate address following the instruction. The MADD and MADS instructions load the PC with BMAR contents.

For a more detailed explanation of how the program address is loaded into the PC, see Section 4.1, *Program Counter*, on page 4-2. See also Section 4.6, *Single Instruction Repeat Function*, on page 4-22, and Chapter 6, *Assembly Language Instructions*, for more information.

4.4 Status and Control Registers

The ’C5x has four status and control registers:

- Circular buffer control register (CBCR) and processor mode status register (PMST) contain status and control information. Since these registers are memory-mapped, they can be stored into and loaded from data memory; therefore, the status of the CPU can be saved and restored for subroutines and interrupt service routines (ISRs).
- Status registers ST0 and ST1 contain the status of various conditions and modes compatible with the ’C2x.

4.4.1 Circular Buffer Control Register (CBCR)

The CBCR resides in the memory-mapped register space of data memory page 0 and can be saved in the same way as any other data memory location. The CBCR can be acted upon directly by the CALU and the PLU. The CALU and the PLU operations change the status register bits during the execution phase of the pipeline. The next two instructions after a status register update must not be affected by the reconfiguration caused by the status update. Table 7–10 on page 7-24 shows the required latencies between instructions and register accesses.

The CBCR bits are shown in Figure 4–2 and defined in Table 4–2.

Do not use the same AR to access both circular buffers or unexpected results will occur.

Figure 4–2. Circular Buffer Control Register (CBCR) Diagram

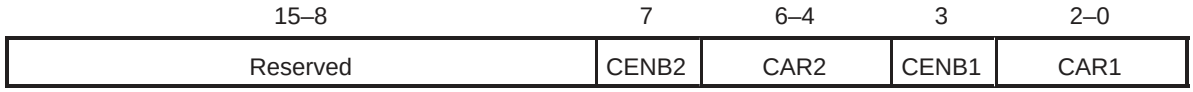


Table 4–2. Circular Buffer Control Register (CBCR) Bit Summary

Bit	Name	Reset value	Function
15–8	Reserved	—	These bits are reserved.
7	CENB2	0	Circular buffer 2 enable bit. This bit enables/disables circular buffer 2.
			CENB2 = 0 Circular buffer 2 is disabled.
			CENB2 = 1 Circular buffer 2 is enabled.
6–4	CAR2	—	Circular buffer 2 auxiliary register bits. These bits select which auxiliary register (AR0–AR7) is assigned to circular buffer 2.
3	CENB1	0	Circular buffer 1 enable bit. This bit enables/disables circular buffer 1.
			CENB1 = 0 Circular buffer 1 is disabled.
			CENB1 = 1 Circular buffer 1 is enabled.
2–0	CAR1	—	Circular buffer 1 auxiliary register bits. These bits select which auxiliary register (AR0–AR7) is assigned to circular buffer 1.

4.4.2 Processor Mode Status Register (PMST)

The PMST resides in the memory-mapped register space of data memory page 0 and can be saved in the same way as any other data memory location. The PMST can be acted upon directly by the CALU and the PLU. The CALU and the PLU operations change the status register bits during the execution phase of the pipeline. The next two instructions after a status register update must not be affected by the reconfiguration caused by the status update.

The PMST has an associated 1-level deep shadow register stack for automatic context-saving when an interrupt trap is taken. The PMST is automatically restored upon a return from interrupt (RETI) or return from interrupt with interrupt enable (RETE) instruction. Table 7–10 on page 7-24 shows the required latencies between instructions and register accesses.

The PMST bits are shown in Figure 4–3 and defined in Table 4–3.

Figure 4–3. Processor Mode Status Register (PMST) Diagram

15–11	10	9	8	7	6	5	4	3	2	1	0
IPTR	0	0	0	AVIS	0	OVLY	RAM	MP/$\overline{MC}$	NDX	TRM	BRAF

Table 4–3. Processor Mode Status Register (PMST) Bit Summary

Bit	Name	Reset value	Function
15–11	IPTR	00000	Interrupt vector pointer bits. These bits select any of 32 2K-word pages where the interrupt vectors reside. The interrupt vectors can be remapped to RAM for boot-loaded operations by loading the IPTR bits. At reset, the IPTR bits are cleared; therefore, the reset vector always resides at address 0h in program memory space.
10–8		000	These bits are read as 0.
7	AVIS	0	Address visibility bit. This bit enables/disables the internal program address to be visible at the address pins. AVIS = 0 The internal program address is driven to the pins so that the address can be traced and the interrupt vector can be decoded in conjunction with $\overline{IACK}$ when the interrupt vectors reside in on-chip memory. AVIS = 1 The address lines do not change with the internal program address. The control and data lines are not affected and the address bus is driven with the last address on the bus.
6		0	This bit is read as 0.
5	OVLY	0	RAM overlay bit. This bit enables/disables the on-chip single-access RAM (SARAM) to be addressable in data memory space. The OVLY bit is used in conjunction with the RAM bit to configure the on-chip SARAM. See Table 4–4 on page 4-10 for specific mappings of the on-chip SARAM. OVLY = 0 The on-chip SARAM is not addressable in data memory space. OVLY = 1 The on-chip SARAM is mapped into data memory space.
4	RAM	0	Program RAM enable bit. This bit enables/disables the on-chip single-access RAM (SARAM) to be addressable in program memory space. The RAM bit is used in conjunction with the OVLY bit to configure the on-chip SARAM. See Table 4–4 on page 4-10 for specific mappings of the on-chip SARAM. RAM = 0 The on-chip SARAM is not addressable in program memory space. RAM = 1 The on-chip SARAM is mapped into program memory space.

† MP/$\overline{MC}$ is the logic level of MP/$\overline{MC}$ pin reset value.

Table 4–3. Processor Mode Status Register (PMST) Bit Summary (Continued)

Bit	Name	Reset value	Function
3	MP/$\overline{MC}$	†	Microprocessor/microcomputer bit. This bit enables/disables the on-chip ROM to be addressable in program memory space. At reset, the MP/$\overline{MC}$ bit is set to the value corresponding to the logic level on the MP/$\overline{MC}$ pin. The level on the MP/$\overline{MC}$ pin is sampled at reset only and can have no effect until the next reset. MP/$\overline{MC}$ = 0 The on-chip ROM is mapped into program memory space. MP/$\overline{MC}$ = 1 The on-chip ROM is not addressable in program memory space.
2	NDX	0	Enable extra index register bit. This bit determines whether a ’C2x-compatible instruction that modifies or writes to auxiliary register 0 (AR0) also modifies or writes to the index register (INDX) and the auxiliary register compare register (ARCR) to maintain ’C5x object-code compatibility with the TMS320C2x. NDX = 0 ’C2x-compatible mode. Any ’C2x-compatible instruction that modifies or writes AR0 also modifies or writes the INDX and ARCR because the ’C2x uses AR0 for indexing and AR compare operations. NDX = 1 ’C5x-enhanced mode. Any ’C2x-compatible instruction does not affect the INDX and ARCR. The ’C2x-compatible instructions affect only AR0 of the ’C5x.
1	TRM	0	Enable multiple TREGs bit. This bit determines whether a ’C2x-compatible instruction that loads TREG0 also loads TREG1 and TREG2 to maintain ’C5x object-code compatibility with the TMS320C2x. TRM = 0 ’C2x-compatible mode. Any ’C2x-compatible instruction that loads TREG0 also loads TREG1 and TREG2 because the ’C2x uses TREG as a shift count for the prescaling shifter and as a bit address in the BITT instruction. TRM = 1 ’C5x-enhanced mode. Any ’C2x-compatible instruction does not load TREG1 and TREG2. The ’C2x-compatible instructions affect only TREG0 of the ’C5x.
0	BRAF	0	Block repeat active flag bit. This bit indicates that a block repeat is currently active. BRAF = 0 The block repeat is deactivated. The BRAF bit is cleared when the block repeat counter register (BRCR) decrements below 0. BRAF = 1 The block repeat is active. The BRAF bit is automatically set when an RPTB instruction is executed.

† MP/$\overline{MC}$ is the logic level of MP/$\overline{MC}$ pin reset value.

Table 4–4. On-Chip RAM Configuration Using OVLY and RAM Bits

Bit values		On-Chip SARAM Configuration
OVLY	RAM	
0	0	Disabled. The on-chip SARAM is not addressable.
0	1	The on-chip SARAM is mapped into program space.
1	0	The on-chip SARAM is mapped into data space.
1	1	The on-chip SARAM is mapped into both program and data spaces.

4.4.3 Status Registers (ST0 and ST1)

The status registers can be stored into data memory and loaded from data memory, thereby allowing the ’C5x status to be saved and restored for subroutines. The LST instruction writes to ST0 and ST1, and the SST instruction reads from them, except that the ARP bits and INTM bit are not affected by the LST #0 instruction. Unlike the PMST and CBCR, the ST0 and ST1 do not reside in the memory map and, therefore, cannot be handled by using the PLU instructions.

The ST0 and ST1 each have an associated 1-level deep shadow register stack for automatic context-saving when an interrupt trap is taken. The registers are automatically restored upon a return from interrupt (RETI) or return from interrupt with interrupt enable (RETE) instruction. Note that the INTM bit in ST0 and the XF bit in ST1 are not saved on the stack or restored from the stack on an automatic context save. This feature allows the XF pin to be toggled in an interrupt service routine and also allows automatic context saves.

The INTM and OVM bits in ST0 and the C, CNF, HM, SXM, TC, and XF bits in ST1 can be individually set using the SETC instruction or individually cleared using the CLRC instruction. For example, the sign-extension mode (SXM) bit is set with SETC SXM or cleared with CLRC SXM. The DP bits in ST0 can be loaded using the LDP instruction. The PM bits in ST1 can be loaded using the SPM instruction.

The ST0 bits are shown in Figure 4–4 and defined in Table 4–5.

Figure 4–4. Status Register 0 (ST0) Diagram

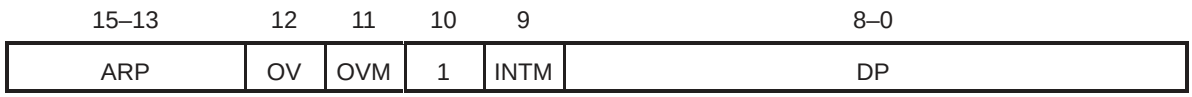


Table 4–5. Status Register 0 (ST0) Bit Summary

Bit	Name	Reset value	Function
15–13	ARP	X	Auxiliary register pointer. These bits select the auxiliary register (AR) to be used in indirect addressing. When the ARP is loaded, the previous ARP value is copied to the auxiliary register buffer (ARB) in ST1. The ARP can be modified by memory-reference instructions when you use indirect addressing, and by the MAR or LST #0 instruction. When an LST #1 instruction is executed, the ARP is loaded with the same value as the ARB.
12	OV	0	Overflow flag bit. This bit indicates that an arithmetic operation overflow in the arithmetic logic unit (ALU). The OV bit can be modified by the LST #0 instruction. OV = 0 Overflow did not occur in the ALU. The OV bit is cleared by a reset or a conditional branch (BCND/BCNDD on OV/NOV). OV = 1 Overflow does occur in the ALU. As a latched overflow signal, the OV bit remains set.
11	OVM	X	Overflow mode bit. This bit enables/disables the accumulator overflow saturation mode in the arithmetic logic unit (ALU). The OVM bit can be modified by the LST #0 instruction. OVM = 0 Disabled. An overflowed result is loaded into the accumulator without modification. The OVM bit can be cleared by the CLRC OVM instruction. OVM = 1 Overflow saturation mode. An overflowed result is loaded into the accumulator with either the most positive (00 7FFF FFFFh) or the most negative value (FF 8000 0000h). The OVM bit can be set by the SETC OVM instruction.
10		1	This bit is read as 1.

Table 4–5. Status Register 0 (ST0) Bit Summary (Continued)

Bit	Name	Reset value	Function
9	INTM	1	Interrupt mode bit. This bit globally masks or enables all interrupts. The INTM bit has no effect on the nonmaskable $\overline{RS}$ and $\overline{NMI}$ interrupts. Note that the INTM bit is unaffected by the TRAP and LST #0 instructions. The INTM bit is not saved on the stack or restored from the stack on an automatic context save during interrupt service routines. INTM = 0 All unmaskable interrupts are enabled. The INTM bit can be cleared by the CLRC INTM or RETE instruction. INTM = 1 All maskable interrupts are disabled. The INTM bit can be set by the SETC INTM or INTR instruction, a $\overline{RS}$ and $\overline{IACK}$ signal, or when a maskable interrupt trap is taken.
8–0	DP	X	Data memory page pointer bits. These bits specify the address of the current data memory page. The DP bits are concatenated with the 7 LSBs of an instruction word to form a direct memory address of 16 bits. The DP bits can be modified by the LST #0 or LDP instruction.

The ST1 bits are shown in Figure 4–5 and defined in Table 4–6.

Figure 4–5. Status Register 1 (ST1) Diagram

15–13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARB	CNF	TC	SXM	C	1	1	HM	1	XF	1	1	PM	

Table 4–6. Status Register 1 (ST1) Bit Summary

Bit	Name	Reset value	Function
15–13	ARB	XXX	Auxiliary register buffer. This 3-bit field holds the previous value contained in the auxiliary register pointer (ARP) in ST0. Whenever the ARP is loaded, the previous ARP value is copied to the ARB, except when using the LST #0 instruction. When the ARB is loaded using the LST #1 instruction, the same value is also copied to the ARP. This is useful when restoring context (when not using the automatic context save) in a subroutine that modifies the current ARP.
12	CNF	0	On-chip RAM configuration control bit. This 1-bit field enables the on-chip dual-access RAM block 0 (DARAM B0) to be addressable in data memory space or program memory space. The CNF bit can be modified by the LST #1 instruction. CNF = 0 The on-chip DARAM block 0 is mapped into data memory space. The CNF bit can be cleared by a reset or the CLRC CNF instruction. CNF = 1 The on-chip DARAM block 0 is mapped into program memory space. The CNF bit can be set by the SETC CNF instruction.
11	TC	X	Test/control flag bit. This 1-bit flag stores the results of the arithmetic logic unit (ALU) or parallel logic unit (PLU) test bit operations. The TC bit is affected by the APL, BIT, BITT, CMPR, CPL, NORM, OPL, and XPL instructions. The status of the TC bit determines if the conditional branch, call, and return instructions execute. The TC bit can be modified by the LST #1 instruction. TC = 0 The TC bit can be cleared by the CLRC TC instruction or any one of the following events: ❑ The result of the logical operation is 1 when tested by the APL, OPL, or XPL instructions. ❑ A bit tested by the BIT or BITT instruction is equal to 0. ❑ A compare condition is false when tested by the CMPR or CPL instruction. ❑ The result of the exclusive-OR operation is false when tested by the NORM instruction.

Table 4–6. Status Register 1 (ST1) Bit Summary (Continued)

Bit	Name	Reset value	Function	
			TC = 1	The TC bit can be set by the SETC TC instruction or any one of the following events: ❑ The result of the logical operation is 0 when tested by the APL, OPL, or XPL instructions. ❑ A bit tested by the BIT or BITT instruction is equal to 1. ❑ A compare condition is true when tested by the CMPR or CPL instruction. ❑ The result of the exclusive-OR operation is true when tested by the NORM instruction.
10	SXM	1	Sign-extension mode bit. This 1-bit field enables/disables sign extension of an arithmetic operation. The SXM bit does not affect the operations of certain arithmetic or logical instructions; the ADDC, ADDS, SUBB, or SUBS instruction suppresses sign extension, regardless of SXM. The SXM bit can be modified by the LST #1 instruction.	
			SXM = 0	Sign extension is suppressed. The SXM bit can be cleared by the CLRC SXM instruction.
			SXM = 1	Sign extension is produced on data as the data is passed into the accumulator through the scaling shifter. The SXM bit can be set by a reset or the SETC SXM instruction.
9	C	1	Carry bit. This 1-bit field indicates an arithmetic operation carry or borrow in the arithmetic logic unit (ALU). The single-bit shift and rotate instructions affect the C bit. The C bit can be modified by the LST #1 instruction.	
			C = 0	The result of a subtraction generates a borrow or the result of an addition (except ADD with a 16-bit shift instruction) did not generate a carry. The ADD with a 16-bit shift instruction can only set the bit (by a carry operation); otherwise, the bit is unaffected. The C bit can be cleared by the CLRC C instruction.
			C = 1	The result of an addition generates a carry or the result of a subtraction (except SUB with a 16-bit shift instruction) did not generate a borrow. The SUB with a 16-bit shift instruction can only clear the bit (by a borrow operation); otherwise, the bit is unaffected. The C bit can be set by a reset or the SETC C instruction.

Table 4–6. Status Register 1 (ST1) Bit Summary (Continued)

Bit	Name	Reset value	Function
8–7		11	These bits are read as 1.
6	HM	1	Hold mode bit. This 1-bit field determines whether the central processing unit (CPU) stops or continues execution when acknowledging an active $\overline{\text{HOLD}}$ signal. The HM bit can be modified by the LST #1 instruction. HM = 0 — The CPU continues execution from on-chip program memory but puts its external interface in the high-impedance state. The HM bit can be cleared by the CLRC HM instruction. HM = 1 — The CPU halts internal execution. The HM bit can be set by a reset or the SETC HM instruction.
5		1	This bit is read as 1.
4	XF	1	XF pin status bit. This 1-bit field determines the level of the external flag (XF) output pin. The XF bit can be modified by the LST #1 instruction. The XF bit is not saved or restored from the stack on an automatic context save during interrupt service routines. XF = 0 — The XF output pin is set to a logic low. The XF bit can be cleared by the CLRC XF instruction. XF = 1 — The XF output pin is set to a logic high. The XF bit can be set by a reset or the SETC XF instruction.
3–2		11	These bits are read as 1.
1–0	PM	00	Product shift mode bits. This 2-bit field determines the product shifter (P-SCALER) mode and shift value for the product register (PREG) output into the arithmetic logic unit (ALU). The PM bits can be set by the SPM or LST #1 instruction. See Table 4–7 for the product shifter modes. The PM shifts also occur when the PREG contents are stored to data memory. The PREG contents remain unchanged during the shifts. See Section 3.2, *Central Arithmetic Logic Unit (CALU)*, on page 3-7 for details.

Table 4–7. Product Shifter Mode as Determined by PM Bits

PM bit values		
Bit 1	**Bit 0**	**P-SCALER mode for PREG output**
0	0	No shift.
0	1	Left-shifted 1 bit; LSB zero-filled.
1	0	Left-shifted 4 bits; 4 LSBs zero-filled.
1	1	Right-shifted 6 bits; sign extended; 6 LSBs lost. The product is always sign extended, regardless of the value of the SXM bit.

4.5 Conditional Operations

In addition to unconditional branches, calls, and returns, the ’C5x has a full complement of conditional branches, calls, and returns. The execution of these instructions is based on the conditions listed in Table 4–8.

Table 4–8. Conditions for Branch, Call, and Return Instructions

Mnemonic	Condition	Description
EQ	ACC = 0	Accumulator equal to 0
NEQ	ACC ≠ 0	Accumulator not equal to 0
LT	ACC < 0	Accumulator less than 0
LEQ	ACC ≤ 0	Accumulator less than or equal to 0
GT	ACC > 0	Accumulator greater than 0
GEQ	ACC ≥ 0	Accumulator greater than or equal to 0
NC	C = 0	Carry bit cleared
C	C = 1	Carry bit set
NOV	OV = 0	No accumulator overflow detected
OV	OV = 1	Accumulator overflow detected
BIO	$\overline{\text{BIO}}$ is low	$\overline{\text{BIO}}$ signal is low
NTC	TC = 0	Test/control flag cleared
TC	TC = 1	Test/control flag set
UNC	none	Unconditional operation

4.5.1 Conditional Branch

The BCND (conditional branch) is a 2-word instruction. The conditions for the branch are not stable until the fourth cycle of the branch instruction pipeline execution, because the previous instruction must have completely executed for the accumulator’s status bits to be accurate. Therefore, following the branch, the pipeline controller stops the decode of instructions until the conditions are valid. If the conditions defined in the operands of the instruction are met, the PC is loaded with the second word and the CPU starts filling the pipeline with instructions at the branch address. Because the pipeline has been flushed, the branch instruction has an effective execution time of four cycles if the branch is taken. If, however, any of the conditions are not met, the pipeline controller allows the next instruction (already fetched) to be decoded. This means that if the branch is not taken, the effective execution time of the branch is two cycles.

4.5.2 Conditional Call

The CC (conditional call) is a 2-word instruction. The CC instruction operates like the BCND except that the PC pointing to the instruction following the CC is pushed onto the stack. Thus, the return (RET) operation can pop the stack to return to the calling sequence. A subroutine or function can have multiple return paths depending on the data being processed.

4.5.3 Conditional Return

The ’C5x supports conditional returns (RETC) to avoid conditionally branching around the return. Example 4–1 shows an overflow-handling subroutine called if the main algorithm causes an overflow condition. During the subroutine, the ACC is checked and, if it is positive, the subroutine returns to the calling sequence. If it is not positive, additional processing is necessary before the return. Note that RETC, like RET, is a 1-word instruction. However, because of the potential PC discontinuity, RETC operates with the same effective execution time as BCND and CC.

Example 4–1. Use of Conditional Returns (RETC Instruction)

```
        CC     OVER_FLOW,OV  ;If overflow,then execute the
        .                    ;overflow-handling routine.
        .
        .
OVER_FLOW                    ;Overflow-handling routine.
        .
        .
        .
        RETC   GEQ           ;If ACC >= 0, then return.
        .
        .
        .
        RET                  ;Return.
```

4.5.4 Multiconditional Instructions

Multiple conditions can be defined in the operands of the conditional instructions. All defined conditions must be met.

The ’C5x includes instructions that test multiple conditions before passing control to another section of the program. These instructions are: BCND, BCNDD, CC, CCD, RETC, RETCD, and XC. These instructions can test the conditions listed in Table 4–8 individually or in combination with other conditions.

You can combine conditions from the following four groups (Table 4–9). You can select up to four conditions; however, each of these conditions must be from different groups. You cannot have two conditions from the same group. For example, you can test EQ and TC at the same time but not NEQ and GEQ. For example:

```
BCND  BRANCH,LT,NOV,TC      ; If ACC < 0, no overflow
                            ; and TC bit set.
```

In this example, LT (ACC < 0), NOV (OV = 0), and TC (TC = 1) conditions must be met for the branch to be taken.

For a description of the condition codes, see Section 4.5, *Conditional Operations*, on page 4-17.

Table 4–9. Groups for Multiconditional Instructions

Group 1	Group 2	Group 3	Group 4
EQ	OV	C	TC
NEQ	NOV	NC	NTC
GT			BIO
LT			
GEQ			
LEQ			

4.5.5 Delayed Conditional Branches, Calls, and Returns

To avoid flushing the pipeline and causing extra cycles, the ’C5x has a full set of delayed conditional branches, calls, and returns. The one 2-word instruction or two 1-word instructions following a delayed instruction are executed while the instructions at and following the branch address are being fetched, thereby giving an effective 2-cycle branch instead of flushing the pipeline. If the instruction following the delayed instruction is 2 words, only that 2-word instruction is executed before the branch is taken.

Conditions tested in the branch are not affected by the instructions following the delayed branch, as shown in Example 4–2 and Example 4–3.

Example 4–2. Use of Conditional Branch (BCND Instruction)

```
OPL    #030h,PMST
BCND   NEW_ADRS,EQ
```

Example 4–3. Use of Delayed Conditional Branch (BCNDD Instruction)

```
BCNDD  NEW_ADRS,EQ
OPL    #030h,PMST
```

The code in Example 4–2 executes in six cycles (two for the OPL and four for the BCND). The code in Example 4–3 executes in four cycles because the two dead cycles following the BCNDD are filled with the OPL instruction. The condition tested on the branch is not affected by the OPL instruction, thereby allowing it to be executed after the branch.

4.5.6 Conditional Execution

In cases where you want the conditional branch to skip over one or two words of code, the branch can be replaced with the execute conditionally (XC) instruction. There are two forms of the XC instruction. One form is the conditional execute of a 1-word instruction (XC 1). The second form is the conditional execute of one 2-word instruction or two 1-word instructions (XC 2). Conditions for XC are the same as for conditional branches, calls, and returns (see Table 4–8 on page 4-17).

Example 4–4 shows a code example for a conditional branch and Example 4–5 shows a code example for a conditional execution.

Example 4–4. Conditional Branch Operation

```
       BCND SUM,NC
       ADD  ONE
SUM    APAC
```

Example 4–5. Use of Conditional Execution (XC Instruction)

```
       XC   1,C
       ADD  ONE
       APAC
```

The code in Example 4–4 executes in six cycles (four for the BCND, one for the ADD, and one for the APAC). The code in Example 4–5 executes in three cycles (one each for the XC, ADD, and APAC). If the condition (C = 1) is met in Example 4–5, the ADD instruction is executed. If the condition is not met, a no operation (NOP) instruction is executed instead of the ADD.

The condition (C = 1) must be stable one full cycle before the XC instruction is executed. This ensures that the decision is made before the instruction following XC is decoded. You should avoid changing the XC test conditions in the 1-word instruction before XC. If no interrupts occur, this instruction has no effect on XC. However, if an interrupt occurs, it can trap between the instruction and XC, thus, affecting the condition before XC is executed.

Example 4–6 and Example 4–7 show cycle dependency for the XC instruction.

Example 4–6. XC Execution with Unstable Condition

```
LACL   #0               ;ACC = 0
ADD    TEMP1            ;ACC = TEMP1
XC     2,EQ             ;If ACC == 0,
SPLK   #0EEEEh,TEMP2    ;then TEMP2 = 0EEEEh
```

Example 4–7. XC Execution with Stable Condition

```
LACL   #0               ;ACC = 0
ADD    #01234h          ;ACC = 00001234
XC     2,EQ             ;If ACC == 0,
SPLK   #0EEEEh,TEMP2    ;then TEMP2 is unmodified
```

In the code in Example 4–6, the NEQ condition (ACC = TEMP1 $\neq$ 0) is not stable one full cycle before the XC instruction is executed. The NEQ status, caused by the ADD instruction, is not established because the ADD is only a 1-cycle instruction. Therefore, the previous EQ condition, caused by the LACL instruction, determines the conditional execute. Since the condition is met (ACC = 0), the one 2-word instruction is executed, and TEMP2 is loaded by the SPLK instruction. If an interrupt occurs, it can trap before XC and after ADD so the SPLK instruction cannot execute. In the code in Example 4–7, the NEQ condition (ACC $\neq$ 0) is stable one full cycle before the XC instruction is executed. The NEQ status, caused by the ADD instruction, is established because the long immediate value (#01234h) used with ADD is a 2-cycle instruction. Since the condition is not met, a NOP instruction is executed instead of the one 2-word instruction, and TEMP2 is not affected. If an interrupt occurs, it has no effect on this instruction sequence.

4.6 Single Instruction Repeat Function

A single instruction can be repeated N + 1 times, where N is the value loaded into a 16-bit repeat counter register (RPTC) by the RPT or RPTZ instruction. The maximum number of executions of a given instruction is 65 536. The RPTC cannot be programmed; it is is cleared by reset and loaded only by the RPT or RPTZ instruction. When the repeat function is used, RPTC is decremented each time the instruction is executed until the RPTC equals 0. Once a repeat instruction is decoded, all interrupts, including $\overline{\text{NMI}}$ (but not $\overline{\text{RS}}$), are masked until the completion of the repeat loop. However, the ’C5x responds to the $\overline{\text{HOLD}}$ signal while executing a repeat loop.

The RPTC is a memory-mapped register. However, you should avoid writing to this register. Writing to this register can cause undesired results.

You can use the repeat function with instructions such as multiply/accumulates, block moves, I/O transfers, and table reads/writes. When you use the repeat function, these multicycle instructions are pipelined and the instruction effectively becomes a single-cycle instruction after the first iteration. Absolute program or data addresses are automatically incremented when you use the repeat function. For example, the TBLR instruction can require three or more cycles to execute, but when the instruction is repeated, a table location can be read every cycle.

Not all instructions can be repeated or are meaningful to repeat. Table 4–10 through Table 4–13 list all ’C5x instructions according to their repeatability.

Table 4–10. Multi-cycle Instructions Transformed Into Single-Cycle Instructions by the Repeat Function

Mnemonic†	Description
BLDD	Block move from data to data memory
BLDP	Block move from data to program memory with destination address in BMAR
BLPD	Block move from program to data memory
IN	Input data from I/O port to data memory location
MAC	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by program memory value and store result in PREG
MACD	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by program memory value and store result in PREG; and move data
MADD	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG; and move data
MADS	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG
OUT	Output data from data memory location to I/O port
TBLR	Transfer data from program to data memory with source address in ACCL
TBLW	Transfer data from data to program memory with destination address in ACCL

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table 4–11. Repeatable Instructions

Mnemonic†	Description
ADCB	Add ACCB and carry bit to ACC
ADD	Add data memory value, with left shift, to ACC
ADDB	Add ACCB to ACC
ADDC	Add data memory value and carry bit to ACC with sign extension suppressed
ADDS	Add data memory value to ACC with sign extension suppressed
ADDT	Add data memory value, with left shift specified by TREG1, to ACC
APAC	Add PREG, with shift specified by PM bits, to ACC
APL	AND data memory value with DBMR, and store result in data memory location
BLDD	Block move from data to data memory
BLDP	Block move from data to program memory with destination address in BMAR
BLPD	Block move from program to data memory
BSAR	Barrel-shift ACC right
DMOV	Move data in data memory
IN	Input data from I/O port to data memory location
LMMR	Load data memory value to memory-mapped register
LTA	Load data memory value to TREG0; add PREG, with shift specified by PM bits, to ACC
LTD	Load data memory value to TREG0; add PREG, with shift specified by PM bits, to ACC; and move data
LTS	Load data memory value to TREG0; subtract PREG, with shift specified by PM bits, from ACC
MAC	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by program memory value and store result in PREG
MACD	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by program memory value and store result in PREG; and move data
MADD	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG; and move data

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table 4–11. Repeatable Instructions (Continued)

Mnemonic†	Description
MADS	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG
MPYA	Add PREG, with shift specified by PM bits, to ACC; multiply data memory value by TREG0 and store result in PREG
MPYS	Subtract PREG, with shift specified by PM bits, from ACC; multiply data memory value by TREG0 and store result in PREG
MAR	Modify ARn
NOP	No operation
NORM	Normalize ACC
OPL	OR data memory value with DBMR and store result in data memory location
OUT	Output data from data memory location to I/O port
POP	Pop top of stack to ACCL; zero ACCH
POPD	Pop top of stack to data memory location
PSHD	Push data memory value to top of stack
PUSH	Push ACCL to top of stack
ROL	Rotate ACC left 1 bit
ROLB	Rotate ACCB and ACC left 1 bit
ROR	Rotate ACC right 1 bit
RORB	Rotate ACCB and ACC right 1 bit
SACH	Store ACCH, with left shift, in data memory location
SACL	Store ACCL, with left shift, in data memory location
SAMM	Store ACCL in memory-mapped register
SAR AR, {ind}	Store ARn (modified in indirect addressing mode) in data memory location
SATH	Barrel-shift ACC right 0 or 16 bits as specified by TREG1
SATL	Barrel-shift ACC right as specified by TREG1
SBB	Subtract ACCB from ACC
SBBB	Subtract ACCB and logical inversion of carry bit from ACC

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table 4–11. Repeatable Instructions (Continued)

Mnemonic†	Description
SFL	Shift ACC left 1 bit
SFLB	Shift ACCB and ACC left 1 bit
SFR	Shift ACC right 1 bit
SFRB	Shift ACCB and ACC right 1 bit
SMMR	Store memory-mapped register in data memory location
SPAC	Subtract PREG, with shift specified by PM bits, from ACC
SPH	Store PREG high byte, with shift specified by PM bits, in data memory location
SPL	Store PREG low byte, with shift specified by PM bits, in data memory location
SQRA	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; square value and store result in PREG
SQRS	Subtract PREG, with shift specified by PM bits, from ACC; load data memory value to TREG0; square value and store result in PREG
SST	Store STn in data memory location
SUB	Subtract data memory value, with left shift, from ACC
SUBB	Subtract data memory value and logical inversion of carry bit from ACC with sign extension suppressed
SUBC	Conditional subtract
SUBS	Subtract data memory value from ACC with sign extension suppressed
SUBT	Subtract data memory value, with left shift specified by TREG1, from ACC
TBLR	Transfer data from program to data memory with source address in ACCL
TBLW	Transfer data from data to program memory with destination address in ACCL
XPL	Exclusive-OR data memory value with DBMR and store result in data memory location

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table 4–12. Instructions Not Meaningful to Repeat

Mnemonic†	Description
ABS	Absolute value of ACC; zero carry bit
AND	AND data memory value with ACCL; zero ACCH
ANDB	AND ACCB with ACC
BIT	Test bit
BITT	Test bit specified by TREG2
CLRC	Clear status bit
CMPL	1s complement ACC
CMPR	Compare ARn with ARCR as specified by CM bits
CPL	Compare data memory value with DBMR
CRGT	Store ACC in ACCB if ACC > ACCB
CRLT	Store ACC in ACCB if ACC < ACCB
EXAR	Exchange ACCB with ACC
LACB	Load ACC to ACCB
LACC	Load data memory value, with left shift, to ACC
LACL	Load data memory value to ACCL; zero ACCH
LACT	Load data memory value, with left shift specified by TREG1, to ACC
LAMM	Load contents of memory-mapped register to ACCL; zero ACCH
LAR	Load data memory value to ARx
LDP	Load data memory value to DP bits
LPH	Load data memory value to PREG high byte
LST	Load data memory value to STm
LT	Load data memory value to TREG0
LTP	Load data memory value to TREG0; store PREG, with shift specified by PM bits, in ACC
MPY	Multiply data memory value by TREG0 and store result in PREG
MPYU	Multiply unsigned data memory value by TREG0 and store result in PREG

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table 4–12. Instructions Not Meaningful to Repeat (Continued)

Mnemonic†	Description
NEG	Negate (2s complement) ACC
OR	OR data memory value with ACCL
ORB	OR ACCB with ACC
PAC	Load PREG, with shift specified by PM bits, to ACC
SACB	Store ACC in ACCB
SAR AR, dma	Store ARn direct addressed in data memory location
SETC	Set status bit
SPM	Set product shift mode (PM) bits
XOR	Exclusive-OR data memory value with ACCL
XORB	Exclusive-OR ACCB with ACC
ZALR	Zero ACCL and load ACCH with rounding
ZAP	Zero ACC and PREG
ZPR	Zero PREG

† **Bold** typeface indicates instructions that are new for the 'C5x instruction set.

Table 4–13. Nonrepeatable Instructions

Mnemonic†	Description
ADD *#k*	Add short immediate to ACC
ADD *#lk, shift*	Add long immediate, with left shift, to ACC
ADRK	Add short immediate to AR
AND *#lk, shift*	AND long immediate, with left shift, with ACC
APL *#lk*	AND data memory value with long immediate and store result in data memory location
B	Branch unconditionally
BACC	Branch to program memory location specified by ACCL
BACCD	Delayed branch to program memory location specified by ACCL
BANZ	Branch to program memory location if AR not zero
BANZD	Delayed branch to program memory location if AR not zero
BCND	Branch conditionally to program memory location
BCNDD	Delayed branch conditionally to program memory location
BD	Delayed branch unconditionally
CALA	Call to subroutine addressed by ACCL
CALAD	Delayed call to subroutine addressed by ACCL
CALL	Call to subroutine unconditionally
CALLD	Delayed call to subroutine unconditionally
CC	Call to subroutine conditionally
CCD	Delayed call to subroutine conditionally
CPL *#lk*	Compare data memory value with long immediate
IDLE	Idle until nonmaskable interrupt or reset
IDLE2	Idle until nonmaskable interrupt or reset — low-power mode
INTR	Software interrupt that branches program control to program memory location
LACC *#lk, shift*	Load long immediate, with left shift, to ACC
LACL *#k*	Load short immediate to ACCL; zero ACCH
LAR *#k*	Load short immediate to ARx
LAR *#lk*	Load long immediate to ARx
LDP *#k*	Load short immediate to DP bits

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table 4–13. Nonrepeatable Instructions (Continued)

Mnemonic†	Description
MPY *#k*	Multiply short immediate by TREG0 and store result in PREG
MPY *#lk*	Multiply long immediate by TREG0 and store result in PREG
NMI	Nonmaskable interrupt and globally disable interrupts (INTM = 1)
OPL *#lk*	OR data memory value with long immediate and store result in data memory location
OR *#lk, shift*	OR long immediate, with left shift, with ACC
RET	Return from subroutine
RETC	Return from subroutine conditionally
RETCD	Delayed return from subroutine conditionally
RETD	Delayed return from subroutine
RETE	Return from interrupt with context switch and globally enable interrupts (INTM = 0)
RETI	Return from interrupt with context switch
RPT	Repeat next instruction specified by data memory value
RPTB	Repeat block of instructions specified by BRCR
RPTZ	Clear ACC and PREG; repeat next instruction specified by long immediate
SBRK	Subtract short immediate from AR
SPLK *#lk*	Store long immediate in data memory location
SUB *#k*	Subtract short immediate from ACC
SUB *#lk, shift*	Subtract long immediate, with left shift, from ACC
TRAP	Software interrupt that branches program control to program memory location 22h
XC	Execute next instruction(s) conditionally
XOR *#lk, shift*	XOR long immediate, with left shift, with ACC
XPL *#lk*	Exclusive-OR data memory value with long immediate and store result in data memory location

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

4.7 Block Repeat Function

A block of instructions can be repeated N + 1 times, where N is the value loaded into a 16-bit block repeat counter register (BRCR) by the RPTB instruction. The maximum number of executions of a given instruction block is 65 536. The block repeat feature provides no-overhead looping for implementation of FOR and DO loops. The block repeat function is controlled by three registers (PASR, PAER, and BRCR) and the block repeat active flag (BRAF) bit in the PMST. You can set or clear the BRAF bit via the PMST.

When the repeat block (RPTB) instruction is executed, it automatically sets the BRAF bit, loads the program address start register (PASR) with the address of the instruction following the RPTB instruction, and loads the program address end register (PAER) with its long immediate operand. The long immediate operand is the address of the instruction following the last instruction in the loop, minus 1. The repeat block must contain at least three instruction words. With each PC update, the PAER is compared to the PC. If they are equal, the BRCR contents are compared to 0. If the BRCR is greater than 0, it is decremented, and the PASR is loaded into the PC, therefore restarting the loop. If they are not equal, the BRAF bit is cleared and the processor resumes execution past the end of the code's loop. Example 4–8 shows how the RPTB instruction can be used.

Example 4–8. Use of Block Repeat (RPTB Instruction)

```
    SPLK   #0Fh,BRCR ;Set loop count to 16.
    RPTB   END_LOOP-1   ;For I = BRCR; I >=0; I--.
*
    ZAP                 ;ACC = PREG = 0.
    SQRA   *,AR2        ;PREG = X^2.
    SPL    SQRX         ;Save X^2.
    MPY    *            ;PREG = b x X.
    LTA    SQRX         ;ACC = bX. TREG = X^2.
    MPY    *            ;PREG = aX^2.
    APAC                ;ACC = aX^2 + bX.
    ADD    *,0,AR3      ;ACC = aX^2 + bX + c = Y.
    SACL   *,0,AR1      ;Save Y.
    CRGT                ;Save MAX.
END_LOOP
```

Example 4–8 implements 16 executions of $Y = aX^2 + bX + c$ and saves the maximum value in ACCB. Note that the initialization of the auxiliary registers is not shown in the coded example. PAER is loaded with the address of the last word in the code segment. The label END_LOOP is placed after the last instruction, and the RPTB instruction long immediate is defined as END_LOOP–1, in case the last word in the loop is a 2-word instruction.

4.7.1 Context Save and Restore Used With Block Repeat

There is only one set of block repeat registers, so multiple block repeats cannot be nested without saving the context of the outside block or using the BANZD instruction. The simplest method of executing nested loops is to use the RPTB instruction for only the innermost loop and using the BANZD instruction for all the outer loops. This is still a valuable cycle-saving operation because the innermost loop is repeated significantly more times than the outer loops. You can nest block repeats by storing the context of the outer loop before initiating the inner loop, then restoring the outer loop's context after the inner loop completes. The context save and restore are shown in Example 4–9.

Example 4–9. Context Save and Restore Used With Block Repeat

```
    SMMR   BRCR,TEMP1       ;Save block repeat counter
    SMMR   PASR,TEMP2       ;Save block start address
    SMMR   PAER,TEMP3       ;Save block end address
    SPLK   #NUM_LOOP,BRCR   ;Set inner loop count
    RPTB   END_INNER        ;For I = 0; I<=BRCR; I++
     .
     .
     .
END_INNER
    LMMR   BRCR,TEMP1       ;Restore block repeat counter
    OPL    #1,PMST          ;Set BRAF to continue outer loop
    LMMR   PASR,TEMP2       ;Restore block start address
    LMMR   PAER,TEMP3       ;Restore block end address
```

In Example 4–9, the context save and restore operations require 14 cycles. Repeated single and BANZ/BANZD loops can also be inside a block repeat and can include subroutine calls. Upon returning from a subroutine call, the block repeat resumes. Repeated blocks can also be interrupted. When an enabled interrupt occurs during a repeated block of code, the CALU traps to the interrupt and, when the interrupt service routine returns, the block repeat resumes.

Caution should be exercised when interrupting block repeats. If the interrupt service routine uses block repeats, check whether a block repeat has been interrupted and, if so, save the context of the block repeat, as shown in Example 4–9. Smaller external loops can be implemented with the BANZD-looping method that requires two extra cycles per loop (that is, if the loop count is less than eight, it can be more efficient to use the BANZD technique). Single-cycle instructions can be repeated within a block repeat by using the RPT or RPTZ instructions.

WHILE loops can be implemented with the RPTB instruction and a conditional reset of the BRAF bit. The following code example clears BRAF bit so that the

processor will drop out of the code loop and continue to sequentially access instructions past the end of the loop if an overflow occurs:

```
XC     2,OV          ;If overflow,
APL    #0FFFEh,PMST ;then turn off block repeat.
```

The equivalent of a WHILE loop can be implemented by clearing the BRAF bit if the exit condition is met. If this is done, the program completes the current pass through the loop but does not go back to the top. To exit, the BRAF bit must be cleared at least four instruction words before the end of the loop. You can exit block repeat loops and return to them without stopping and restarting the loop. Branches, calls, and interrupts do not necessarily affect the loop. When program control is returned to the loop, loop execution is resumed. Example 4–10 shows the block repeat with a small loop of code that executes a series of tasks. The tasks are stored in a table addressed by TEMP0F. The number of tasks to be executed is defined at NUM_TASKS.

Example 4–10. Block Repeat with Small Loop of Code

```
    BLPD   NUM_TASKS,BRCR       ;Set loop count.
    SPLK   #(TASKS-1),TEMP0F    ;TEMP0F points to list of
                                ;tasks.
    RPTB   ENDCALL-1            ;For I = 0, I <= NUM_TASKS;
                                ;I++.
TASK_HANDLER
    LACC   TEMP0F               ;ACC points to task table.
    ADD    #1                   ;Increment pointer to next
                                ;task.
    SACL   TEMP0F               ;Save for next pass of loop.
    TBLR   TEMP0E               ;Get task address.
    LACC   TEMP0E               ;ACC = task address.
    CALA                        ;Call task.
ENDCALL
```

In the setup of Example 4–10, the BRCR is loaded with the number of tasks to be executed. Next, the address of the task table is loaded into a temporary register. The block repeat is started with the execution of the RPTB instruction. The PASR is loaded with the address of the LACC TEMP0F instruction. The PAER is loaded with the address of the last word of code. Notice that the label marking the end of the loop is placed after the last instruction, then the PAER is loaded with that label, minus 1. It is possible to place the label before the CALA instruction, then load the PAER with the label address because this is a 1-word instruction. However, if the last instruction in this loop had been a 2-word instruction, the second word of the instruction would not be read and the long immediate operand would be substituted with the first instruction in the loop.

Inside the loop, the pointer to the task table is incremented and saved. Then, the task address is read from the table and loaded into the ACC. Next, the task is called by the CALA instruction. Notice that, when the task returns to the task handler, it returns to the top of the loop. This is because the PC has already been loaded with the PASR before the CALA executes the PC discontinuity. Therefore, when the CALA is executed, the address at the top of the loop is pushed onto the PC stack.

4.7.2 Interrupt Operation in a Block Repeat

The single-word instruction at the end of a repeat block is not interruptible, except, when the previous instruction is a single-word multiple cycles instruction as shown in Example 4–11 and Example 4–12. Since BLDD BMAR, *+ is a single-word multiple-cycle instruction, the interrupt return is to the end of the repeat block (see Example 4–12).

An incoming interrupt is latched by the ’C5x as soon as it meets the interrupt timing requirement. However, the PC does not branch to the corresponding interrupt service routine vector if it is fetching the last word of a repeat block loop. This is the functional equivalent to disabling interrupts before the last instruction word is fetched and reenabling interrupts afterward. Interrupt operation with repeat blocks can potentially increase the worst-case interrupt latency time.

Note:

When the case in Example 4–12 occurs, execute the following steps:

1) Save the PMST at the beginning of the interrupt service routine.
2) Clear the BRAF bit inside the interrupt service routine.
3) Restore the PMST before returning from the interrupt service routine.

Example 4–11. Interrupt Operation With a Single-Word Instruction at the End of an RPTB

```
    RPTB   END_LOOP-1
    SAR AR0,*  ←  return from interrupt here if not the last loop iteration
    .
    .
    .
    LACC   *+
    SACL   *   ←  interrupt occurs here
ENDLOOP:
    MAR *,AR1  ←  return from interrupt here if interrupt occurs during last
                  two instruction words of the last loop iteration
```

Example 4–12. Interrupt Operation With a Single-Word Instruction Before the End of RPTB

```
    RPTB   END_LOOP-1
    SAR AR0,*
    .
    .
    .
    BLDD   BMAR,*+
                        ←Interrupt occurs here and  return at SACL
    SACL   *
END_LOOP:
    MAR *,AR1
```

4.8 Interrupts

The ’C5x CPU supports 16 user-maskable interrupts ($\overline{\text{INT16}}$–$\overline{\text{INT1}}$); however, each ’C5x DSP does not necessarily use all 16 interrupts. For example, all the ’C5x DSPs use only 9 of these interrupts except ’C57, which uses 10 of them (the others are tied high internally). External interrupts are generated by external hardware using $\overline{\text{INT1}}$–$\overline{\text{INT4}}$. Internal interrupts generated by the on-chip peripherals are:

- The timer (TINT)
- The serial ports (RINT, XINT, TRNT, TXNT, BRNT, and BXNT)
- Host port interface (HINT)

In addition, the ’C5x has three software interrupt instructions, INTR, NMI, and TRAP; and two external nonmaskable interrupts, $\overline{\text{RS}}$ and $\overline{\text{NMI}}$. The reset ($\overline{\text{RS}}$) interrupt has the highest priority, and the $\overline{\text{INT16}}$ interrupt has the lowest priority. The $\overline{\text{INT1}}$–$\overline{\text{INT4}}$ and $\overline{\text{NMI}}$ interrupts are valid if the signal is high for at least two machine cycles and low for a minimum of three machine cycles. This triggering gives the ’C5x the ability to avoid false interrupts from noise or taking multiple interrupts on a single, long interrupt signal.

Note:

If the CPU is in IDLE2 mode, an interrupt input must be high for at least four machine cycles and low for a minimum of five machine cycles to be properly recognized.

4.8.1 Interrupt Vector Locations

Table 4–14 shows interrupt vector locations and priorities for all internal and external interrupts. Interrupt addresses are spaced two locations apart so that branch instructions can be accommodated in these locations. The TRAP instruction (software interrupt) is not prioritized but is included here because it has its own vector location.

To make vectors stored in ROM reprogrammable, you can use the following code:

```
LAMM TEMP0      ;ACC = ISR address.
BACC            ;Branch to ISR.
```

TEMP0 resides in DARAM block B2 and holds the address of the interrupt service routine (ISR). Note that the ISR addresses must be loaded into block B2 before interrupts are enabled. For further information regarding interrupt operation with respect to specific DSPs in the ’C5x generation, see subsection 9.1.2, *External Interrupts*, on page 9-4.

Table 4–14. Interrupt Vector Locations and Priorities

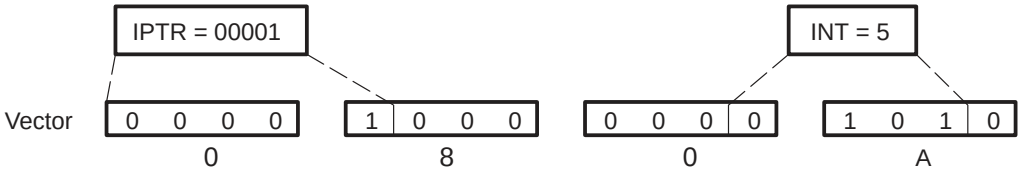
Name	Location		Priority	Function
	Dec	Hex		
$\overline{RS}$	0	0	1 (highest)	External nonmaskable reset signal
$\overline{INT1}$	2	2	3	External user interrupt #1
$\overline{INT2}$	4	4	4	External user interrupt #2
$\overline{INT3}$	6	6	5	External user interrupt #3
TINT	8	8	6	Internal timer interrupt
RINT	10	A	7	Serial port receive interrupt
XINT	12	C	8	Serial port transmit interrupt
TRNT†	14	E	9	TDM port receive interrupt
TXNT‡	16	10	10	TDM port transmit interrupt
$\overline{INT4}$	18	12	11	External user interrupt #4
——	20–23	14–17	N/A	Reserved
HINT	24	18	——	HINT (’C57 only)
——	26–33	1A–21	N/A	Reserved
TRAP	34	22	N/A	Software trap instruction
$\overline{NMI}$	36	24	2	Nonmaskable interrupt
——	38–39	26–27	N/A	Reserved for emulation and test
——	40–63	28–3F	N/A	Software interrupts

† RINT2 on ’C52; BRNT on ’C56/C57
‡ XINT2 on ’C52; BXNT on ’C56/C57

The interrupt vectors can be remapped to the beginning of any 2K-word page in program memory space. The interrupt vector address is generated by concatenating the IPTR bits in the PMST (see subsection 4.4.2, *Processor Mode Status Registers (PMST)*, on page 4-7) with the interrupt vector number (1–16) shifted by 1 as shown in Figure 4–6.

Upon reset, the IPTR bits are all cleared, thereby mapping the vectors to page 0 in program memory space. Therefore, the reset vector always resides at location 0h in program memory space. You can move the interrupt vectors to another location by loading a nonzero value into the IPTR bits. For example, you can move the interrupt vectors of INT 5 (as shown in Figure 4–6) to location 080Ah by loading the IPTR with 1.

Figure 4–6. Interrupt Vector Address Generation



4.8.2 Interrupt Operation

When an interrupt occurs, a flag is activated in the 16-bit interrupt flag register (IFR). The interrupt flag is activated whether the interrupt is enabled or disabled. An interrupt flag (other than from an active serial port) is automatically cleared when the corresponding interrupt trap is taken.

The number of the specific interrupt being acknowledged is indicated by address bits A5–A1 on the falling edge of the interrupt acknowledge ($\overline{\text{IACK}}$) signal. If the interrupt vectors reside in on-chip memory, the CPU should operate in address visibility mode (AVIS = 0) so the interrupt number can be decoded. If an interrupt occurs while the CPU is on hold and HM = 0, the address will not be present when the $\overline{\text{IACK}}$ is activated.

Upon receiving an interrupt, the following actions occur:

- The CPU completes execution of current instruction.
- Interrupts are globally disabled (INTM = 1).
- The PC is pushed to the top of the stack (TOS).
- The PC is set to the interrupt vector address.
- Key registers are saved into context shadow registers.
- $\overline{\text{IACK}}$ signal goes low.
- Corresponding interrupt flag bit in the IFR is cleared.

The ’C5x recognizes pending interrupts on a priority basis. At the start of each machine cycle (when INTM = 0), the interrupt status is polled and the highest priority interrupt present and enabled is executed. When an interrupt is being serviced, even higher priority interrupts cannot be serviced until interrupts are reenabled — usually at the end of the ISR.

4.8.3 Interrupt Flag Register (IFR)

The IFR is a memory-mapped CPU register located at address 06h in data memory space. The IFR can be read to identify pending external and internal interrupts and written to clear interrupts. An interrupt sets its corresponding interrupt flag in the IFR until the interrupt is recognized by the CPU. Any one of the following events clears the interrupt flag:

- The ’C5x is reset ($\overline{RS}$ is active).
- An interrupt trap is taken.
- A 1 is written to the appropriate bit in the IFR.

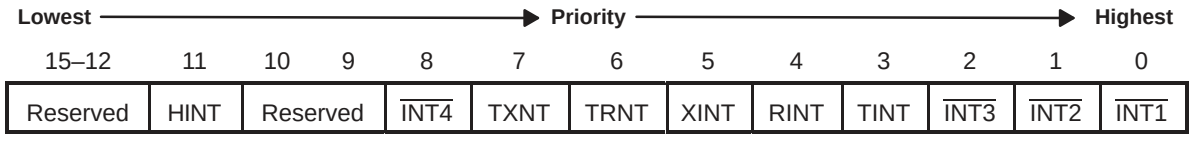
Note that when interrupts are disabled (INTM = 1) and an interrupt causes an IDLE or IDLE2 instruction to be exited, none of the IFR bits are cleared (including the IFR bit that caused the IDLE or IDLE2 to be exited). The only event, other than reset or clearing the IFR bits directly in software, that can cause an IFR bit to be cleared is actually taking the interrupt trap when the the ISR is entered. Therefore, if an interrupt causes an IDLE or IDLE2 instruction to be exited when interrupts are disabled, the corresponding IFR bit is not cleared; whereas, if interrupts are enabled and the ISR is entered, the IFR bit is cleared. Figure 4–7 shows the IFR fields.

A value of 1 in an IFR bit indicates a pending interrupt. A 1 can be written to a specific bit to clear the corresponding interrupt. Writing a 0 to a specific bit has no effect. All pending interrupts can be cleared by writing the current contents of the IFR back into the IFR. The following example clears two interrupts, $\overline{INT1}$ and $\overline{INT3}$, without affecting any other flags that have been set:

```
SPLK #5,IFR  ;Clear flags for INT1 and INT3.
```

The IFR sets only one flag for each interrupt recognized. If several hardware interrupts occur on the same pin before the interrupt is recognized by the CPU, the CPU will respond as though only a single interrupt (the last one) had occurred.

Figure 4–7. Interrupt Flag Register (IFR) Diagram



4.8.4 Interrupt Mask Register (IMR)

The IMR is a memory-mapped CPU register located at address 04h in data memory space. The IMR is used for masking external and internal interrupts. Neither $\overline{\text{NMI}}$ nor $\overline{\text{RS}}$ are in the IMR; therefore, the IMR has no effect on these nonmaskable interrupts.

Figure 4–8 shows the IMR fields.

Figure 4–8. Interrupt Mask Register (IMR) Diagram

15–12	11	10 9	8	7	6	5	4	3	2	1	0
Reserved	HINT	Reserved	$\overline{\text{INT4}}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{\text{INT3}}$	$\overline{\text{INT2}}$	$\overline{\text{INT1}}$

A value of 1 in an IMR bit enables the corresponding interrupt, provided that the INTM bit in ST0 (see subsection 4.4.3, *Status Registers (ST0 and ST1)*, on page 4-10) is cleared. The IMR is accessible with both read and write operations.

4.8.5 Interrupt Mode (INTM) Bit

The INTM bit in ST0 (see subsection 4.4.3, *Status Registers (ST0 and ST1)*, on page 4-10) globally enables or disables all maskable interrupts:

- When INTM = 0, all unmasked interrupts are enabled.
- When INTM = 1, all unmasked interrupts are disabled.

The INTM bit does not modify the IFR or IMR. Any one of the following events sets the INTM bit:

- The ’C5x is reset ($\overline{\text{RS}}$ is active).
- An interrupt trap is taken.
- The NMI instruction is executed.
- The SETC INTM instruction is executed.

Any one of the following events clears the INTM bit:

- The CLRC INTM instruction is executed.
- The RETE instruction is executed.

4.8.6 Nonmaskable Interrupts

The two nonmaskable interrupts, $\overline{RS}$ and $\overline{NMI}$, are unaffected by either the INTM bit or the contents of the IMR. You can use the $\overline{NMI}$ as a soft reset of the ’C5x or as the input to a system’s most time-critical interrupt event. When used as a soft reset, $\overline{NMI}$ does not perform any of the control bit or register initializations that are provided by the $\overline{RS}$ function. The $\overline{NMI}$ trap can be initiated via software using the NMI instruction.

Upon receiving an $\overline{NMI}$, the following actions occur:

1) The CPU completes execution of all instructions in the pipeline.

2) Interrupts are globally disabled (INTM = 1).

3) The PC is set to the $\overline{NMI}$ interrupt vector (location 24h).

Because it is possible to service an $\overline{NMI}$, even during an ISR, the key registers are not saved automatically. The $\overline{NMI}$ is different from $\overline{RS}$ because it does not affect any of the ’C5x modes. The $\overline{NMI}$ is delayed by multicycle instructions (including RPT) and by $\overline{HOLD}$, as described in subsection 4.8.9, *Interrupt Latency*, on page 4-43. $\overline{RS}$ is discussed in Section 4.9, *Reset*, on page 4-45.

4.8.7 Software-Initiated Interrupts

Not all of the 16 CPU interrupts are utilized on any given ’C5x DSP. The vectors for the interrupts that are not tied to specific external pins or internal peripherals can be used as software interrupts. The three software interrupt instructions, INTR, NMI, and TRAP, are unaffected by either the INTM bit or the contents of the IMR. These instructions allow interrupts to be invoked under software control.

The INTR instruction (page 6-111) allows any ISR to be executed from your software. An INTR interrupt for the external interrupts ($\overline{INT1}$–$\overline{INT4}$) executes like an external interrupt described in subsection 4.8.2, *Interrupt Operation*.

The NMI instruction (page 6-179) has the same affect as a hardware nonmaskable interrupt ($\overline{NMI}$). The NMI instruction transfers program control to program memory location 24h. Interrupts are globally disabled (INTM = 1), but key registers are not saved into context shadow registers.

The TRAP instruction (page 6-277) transfers program control to program memory location 22h. The TRAP instruction disables interrupts (INTM = 1), but key registers are not saved into context shadow registers.

4.8.8 Interrupt Context Save

When an interrupt is executed, certain key CPU registers are saved automatically. The PC is saved on an 8-deep hardware stack (see Section 4.2, *Hardware Stack*), which is also used for subroutine calls. Therefore, the CPU supports subroutine calls within an ISR as long as the 8-level stack is not exceeded. Also, there is a 1-deep stack (or shadow registers) for each of the following registers:

- ❑ Accumulator (ACC)
- ❑ Accumulator buffer (ACCB)
- ❑ Auxiliary register compare register (ARCR)
- ❑ Index register (INDX)
- ❑ Processor mode status register (PMST)
- ❑ Product register (PREG)
- ❑ Status register 0 (ST0)
- ❑ Status register 1 (ST1)
- ❑ Temporary register 0 (TREG0) for multiplier
- ❑ Temporary register 1 (TREG1) for shift count
- ❑ Temporary register 2 (TREG2) for bit test

When the interrupt trap is taken, the contents of all these registers are pushed onto a 1-level stack, with the exception of the the INTM bit in ST0 and the XF bit in ST1. On an interrupt, the INTM bit is always set to disable interrupts. The values in the registers at the time of the interrupt trap are still available to the ISR but are also protected in the shadow registers. The shadow registers are copied back to the CPU registers when the RETI or RETE instruction is executed. This function allows the CPU to be used for the ISR without requiring context save and restore overhead in the ISR.

With only a 1-level stack for the registers, nested interrupts cannot be supported. In most cases this is not a problem, because without the context save and restore overhead, serial processing of the interrupts is so efficient that nested interrupt handling is less effective. If the application requires nested interrupts, they can be handled by using a software stack. Software compatibility with the ’C2x is maintained because the RET instruction, if it is used to return from the ISR on a ’C2x, cannot restore these registers. Interrupts are not enabled unless a RETE or CLRC INTM instruction is executed.

In the case where the ISR needs to modify values in these registers with respect to the interrupted code, these registers can be restored from the stack and modified as shown in Example 4–13.

Example 4–13. Modifying Register Values During Interrupt Context Save

```
ISR
   LACC #ISR_RE_ENTER ;ACC = address of reentry point.
   PUSH               ;Top of stack = reentry point.
   RETI               ;Pop all the stacks.
ISR_RE_ENTER
   .
   .
   .
   CLRC INTM
   RET                ;Return to interrupted code.
```

In Example 4–13, the address of the re-entry point within the ISR is pushed onto the PC stack. The RETI instruction pops all the stacks, including the PC stack, and resumes execution. At the end of the ISR, a standard return is executed because the stack is already popped.

4.8.9 Interrupt Latency

The interrupt latency of the ’C5x depends on the current contents of the pipeline. The CPU always completes all instructions in the pipeline before executing a software vector. Figure 4–9 shows the minimum latency from the time an interrupt occurs externally to the $\overline{\text{IACK}}$. The minimum $\overline{\text{IACK}}$ time is defined as eight cycles:

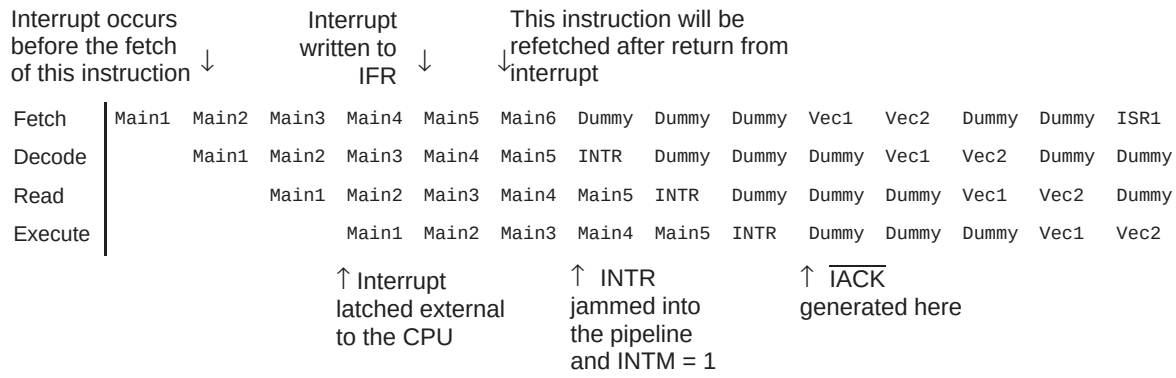
- ❏ 3 cycles to externally synchronize the interrupt
- ❏ 1 cycle for the interrupt to be recognized by the CPU
- ❏ 4 cycles to execute the INTR instruction and flush the pipeline

On the sixth cycle, an INTR is jammed into the pipeline and the INTM bit is set to 1. On the ninth cycle, the interrupt vector is fetched and the $\overline{\text{IACK}}$ signal is generated.

Note that if the instruction immediately ahead of the INTR in the pipeline (Main5 in Figure 4–9) is an SST #0 and INTM was previously cleared, INTM gets set before this instruction executes and INTM is stored as a 1. Therefore, if ST0 is restored in order to return to the previous context, interrupts will be disabled (INTM = 1) rather than enabled.

Accordingly, if this is critical in an application, an SST #0 instruction should be executed only with interrupts disabled or interrupts should be reenabled after loading ST0, if necessary.

Figure 4–9. Minimum Interrupt Latency



The maximum latency is a function of the contents of the pipeline. Multicycle instructions add additional cycles to empty the pipeline. This applies to instructions that are extended via wait-state insertion on memory accesses. The wait states required for interrupt vector accesses also affect the latency.

The repeat instructions (RPT and RPTZ) delay execution of interrupts (including $\overline{\text{NMI}}$, but not $\overline{\text{RS}}$). The repeat instructions require that all executions of the next instruction be completed before allowing an interrupt to execute to protect the context of the repeated instructions. This protection is necessary, because these instructions run parallel operations in the pipeline, and the context of these additional parallel operations cannot be saved in the ISR.

The $\overline{\text{HOLD}}$ function takes precedence over interrupts and can delay the interrupt trap. If an interrupt occurs when the CPU is in hold ($\overline{\text{HOLD}}$ asserted), the interrupt is not taken until $\overline{\text{HOLDA}}$ is deasserted when the hold state ends. However, if the CPU is in the concurrent hold mode (HM = 0) and the interrupt vector table is located in on-chip memory, the CPU takes the interrupt regardless of the $\overline{\text{HOLD}}$ status.

Interrupts cannot be processed between the CLRC INTM instruction and the next instruction in a program sequence. If an interrupt occurs during the decode phase of the CLRC INTM instruction, the CPU always completes CLRC INTM and the following instruction before the pending interrupt is processed. Waiting for these instructions to complete, ensures that a return (RET) can be executed in an ISR before the next interrupt is processed to protect against PC stack overflow. If the ISR is ends with an RETE instruction, the CLRC INTM instruction is unnecessary. Similarly, the SETC INTM instruction and the next instruction cannot be interrupted.

4.9 Reset

Reset ($\overline{RS}$) is a nonmaskable external interrupt that can be used at any time to place the ’C5x into a known state. Reset is typically applied after power-up when the ’C5x is in an unknown state. The reset signal aborts memory operations; therefore, the system should be reinitialized after each reset. Reset is the highest priority interrupt; thus, no other interrupt takes precedence over a reset. You can use the $\overline{NMI}$ interrupt for soft resets because the $\overline{NMI}$ does not abort memory operations or initialize status bits.

A hardware reset clears all pending interrupt flags.

Driving the $\overline{RS}$ signal low causes the ’C5x to terminate execution and forces the PC to the reset vector location 0h in program memory space. At power-up, the state of the ’C5x is undefined. For correct system operation after power-up, the $\overline{RS}$ signal must be asserted low for a minimum of six clock cycles so that the data lines are placed into the high-impedance state and the address lines are driven low. The ’C5x latches the reset pulse and generates an internal reset pulse long enough to guarantee a reset. After the $\overline{RS}$ signal is high for 17 clock cycles, CPU execution begins at location 0h, which normally contains a branch instruction to the system initialization routine. When the ’C5x receives a reset signal, the following sequence of actions occur:

1) The program currently being executed is asynchronously aborted.

2) The CPU registers’ status bits are set per Table 4–15. Note that any remaining status bits remain undefined and should be initialized appropriately.

3) The PC is cleared. The address bus is unknown while $\overline{RS}$ is low. If $\overline{HOLD}$ is asserted while $\overline{RS}$ is low, $\overline{HOLDA}$ is generated. In this case, the address lines are placed into a high-impedance state until $\overline{HOLD}$ is brought back high.

4) A synchronized reset ($\overline{SRESET}$) signal is sent to the peripheral circuits to initialize them. The peripheral registers’ status bits are set per Table 4–16 on page 4-47. See subsection 9.1.3, *Peripheral Reset*, on page 9-6.

Execution starts from program memory location 0h when the $\overline{RS}$ signal is driven high. If $\overline{HOLD}$ is asserted while $\overline{RS}$ is low, normal reset operation occurs internally, but all buses and control lines remain in a high-impedance state, and $\overline{HOLDA}$ is asserted, as shown in Figure 4–10(a) and (b) on page 4-49. However, if $\overline{RS}$ is asserted while $\overline{HOLD}$ and $\overline{HOLDA}$ are low, the CPU comes out of the hold mode momentarily by deactivating $\overline{HOLDA}$. This condition should be avoided. Upon release of $\overline{HOLD}$ and $\overline{RS}$, execution starts from location 0h.

Note that the external parallel interface signals are asynchronously disabled during reset; therefore, external DMA is not supported during reset. See subsection 8.6.2, *External DMA*, on page 8-24 for more information.

Table 4–15. CPU Registers' Bit Status at Reset

Register	Bit Status	Result
ST0	INTM → 1	All maskable interrupts are disabled. Note that $\overline{RS}$ and $\overline{NMI}$ are nonmaskable.
	OV → 0	Overflow bit is cleared.
ST1	C → 1	Carry bit is set.
	CNF → 0	DARAM block B0 is mapped into data memory space.
	HM → 1	Processor halts execution during $\overline{HOLD}$.
	PM → 0	PREG output is not shifted.
	SXM → 1	Sign extension on data is enabled.
	XF → 1	External flag pin is set high.
PMST	AVIS → 0	Internal program address appears at address pins.
	BRAF → 0	Block repeat is disabled.
	IPTR → 0	Reset vector is cleared.
	MP/$\overline{MC}$ → (pin)	MP/$\overline{MC}$ pin is sampled to determine use of on-chip ROM.
	NDX → 0	'C2x-compatible mode is selected.
	OVLY → 0	SARAM block is not mapped to data memory space.
	RAM → 0	SARAM block is not mapped to program memory space.
	TRM → 0	'C2x-compatible mode is selected.
IFR	All bits → 0	No interrupts are pending.
CBCR	CENB1 → 0	Circular buffer 1 is disabled.
	CENB2 → 0	Circular buffer 2 is disabled.
GREG	All bits → 0	All data memory space is configured as local.
RPTC	All bits → 0	Repeat counter is cleared.

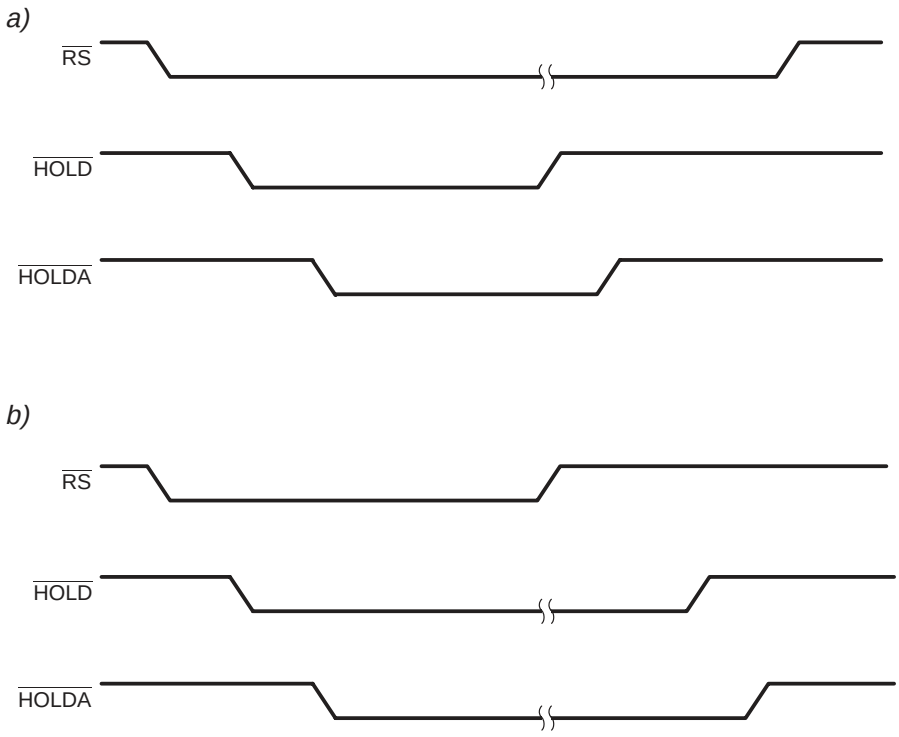
Table 4–16. Peripheral Registers' Bit Status at Reset

Register	Bit Status		Result
PDWSR	All bits	→ 1	All program and data wait-state registers are set to 7.
IOWSR	All bits	→ 1	All I/O wait-state registers are set to 7.
CWSR	BIG	→ 0	I/O space is divided into eight 8K-word blocks.
	D	→ 1	Wait states are enabled for data memory space.
	I/O High	→ 1	Wait states are enabled for upper half of I/O space.
	I/O Low	→ 1	Wait states are enabled for lower half of I/O space.
	P	→ 1	Wait states are enabled for program memory space.
DRR	All bits	→ 0	Data receive register is cleared.
DXR	All bits	→ 0	Data transmit register is cleared.
SPC/BSPC/ TSPC	DLB	→ 0	Digital loop back is disabled.
	FO	→ 0	Data is transmitted/received as 16-bit words.
	Free	→ 0	Stop serial clock is enabled.
	FSM	→ 0	Serial port is operated in continuous mode.
	IN0	→ (pin)	IN0 reflects the current level of the CLKR pin.
	IN1	→ (pin)	IN1 reflects the current level of the CLKX pin.
	MCM	→ 0	CLKX pin is configured as input pin.
	RRDY	→ 0	Receive ready is reset.
	$\overline{\text{RRST}}$	→ 0	Receive serial port is reset.
	RSRFULL	→ 0	SPC only: receive shift register full flag is reset.
	Soft	→ 0	Stop serial clock immediately is enabled.
	TDM	→ 0	TSPC only: TDM port is configured as standard serial port.
	TXM	→ 0	FSX pin is configured as input pin.
	XRDY	→ 1	Transmit ready is reset.
	$\overline{\text{XRST}}$	→ 0	Transmit serial port is reset.
	$\overline{\text{XSREMPTY}}$	→ 0	SPC only: transmit shift register empty flag is reset.

Table 4–16. Peripheral Registers' Bit Status at Reset (Continued)

Register	Bit Status		Result
SPCE	BRE	→ 0	Autobuffering receive is disabled.
	BXE	→ 0	Autobuffering transmit is disabled.
	CLKDV	→ 00011	Internal transmit clock division factor is set to 3.
	CLKP	→ 0	Data is sampled by the receiver on CLKR's falling edge and sent by the transmitter on CLKX's rising edge.
	FE	→ 0	Data is transmitted/received as 16-bit words.
	FIG	→ 0	The frame pulses following first frame restart the serial port interface.
	FSP	→ 0	Frame sync pulses are active high.
	HALTR	→ 0	Autobuffering halt receive is reset.
	HALTX	→ 0	Autobuffering halt transmit is reset.
	PCM	→ 0	Pulse coded modulation is not active.
	RH	→ 0	Receive buffer half received bit is reset.
	XH	→ 0	Transmit buffer half transmitted bit is reset.
TIM	All bits	→ 1	Timer counts down from FFFFh.
PRD	All bits	→ 1	Timer is disabled.
TCR	TDDR	→ 0	Each cycle decrements timer by 1.
	TSS	→ 0	Timer is in run mode.
HPIC	SMOD	0→ 1	Zero while in reset, set to one when reset goes high.
	HINT	→ 0	No interrupt (external $\overline{\text{HINT}}$ pin is high)

Figure 4–10. $\overline{RS}$ and $\overline{HOLD}$ Interaction



4.10 Power-Down Mode

In the power-down mode, the ’C5x enters a dormant state and dissipates less power than in the normal mode. You can invoke the power-down mode by executing either the IDLE or IDLE2 instruction, or by driving the $\overline{\text{HOLD}}$ input low with the HM status bit set. While the ’C5x is in power-down mode, all its internal contents are maintained; this allows operations to continue unaltered when the power-down mode is terminated.

4.10.1 IDLE Instruction

The IDLE instruction halts all CPU activities except the system clock. Since the system clock remains applied to the peripherals, the peripheral circuits continue operating and the CLKOUT1 pin remains active. Thus peripherals such as serial ports and timers can take the CPU out of its power-down state.

This power-down mode is terminated upon receipt of an interrupt. If INTM = 0 when the interrupt takes place, then the CPU enters the ISR when IDLE is terminated. If INTM = 1, then the CPU continues with the instruction following the IDLE.

4.10.2 IDLE2 Instruction

The IDLE2 instruction halts all CPU activities and the on-chip peripherals. Unlike the IDLE instruction, the IDLE2 instruction disables the CLKOUT1 signal. Because the on-chip peripherals are stopped in this power-down mode, they cannot be used to generate the interrupt to wake up the CPU as in the IDLE mode. However, the power is significantly reduced because the complete DSP is stopped. Note that the HPI has some special IDLE2 considerations, see Section 9.10, *Host Port Interface*, on page 9-87.

This power-down mode is terminated by activating any of the external interrupt pins ($\overline{\text{RS}}$, $\overline{\text{NMI}}$, $\overline{\text{INT1}}$, $\overline{\text{INT2}}$, $\overline{\text{INT3}}$, and $\overline{\text{INT4}}$) for at least five machine cycles. If INTM = 0 when the interrupt takes place, then the CPU enters the ISR when IDLE2 is terminated. If INTM = 1, then the CPU continues with the instruction following the IDLE2. Reset all peripherals when IDLE2 terminates, especially if the peripherals are externally clocked.

4.10.3 Power Down Using $\overline{\text{HOLD}}$

The power-down mode can also be initiated by the $\overline{\text{HOLD}}$ signal. When the $\overline{\text{HOLD}}$ signal initiates power-down and HM = 1, the CPU stops executing and address, data, and control lines go into high impedance for further power reduction. When the $\overline{\text{HOLD}}$ signal initiates power-down and HM = 0, the address, data, and control lines go into high impedance, but the CPU continues to execute internally. When external memory accesses are not required in the system, the HM = 0 mode can be used. The ’C5x continues to operate normally unless an off-chip access is required by an instruction, then the CPU halts until the hold is removed.

This power-down mode is terminated when the $\overline{\text{HOLD}}$ signal becomes inactive. $\overline{\text{HOLD}}$ does not stop the operation of on-chip peripherals (serial ports and timers); the peripherals continue to operate regardless of the level on $\overline{\text{HOLD}}$ or the status of the HM bit.

Chapter 5

Addressing Modes

This chapter describes each of the following addressing modes and gives the opcode formats and some examples.

- ❑ Direct addressing
- ❑ Indirect addressing
- ❑ Immediate addressing
- ❑ Dedicated-register addressing
- ❑ Memory-mapped register addressing
- ❑ Circular addressing

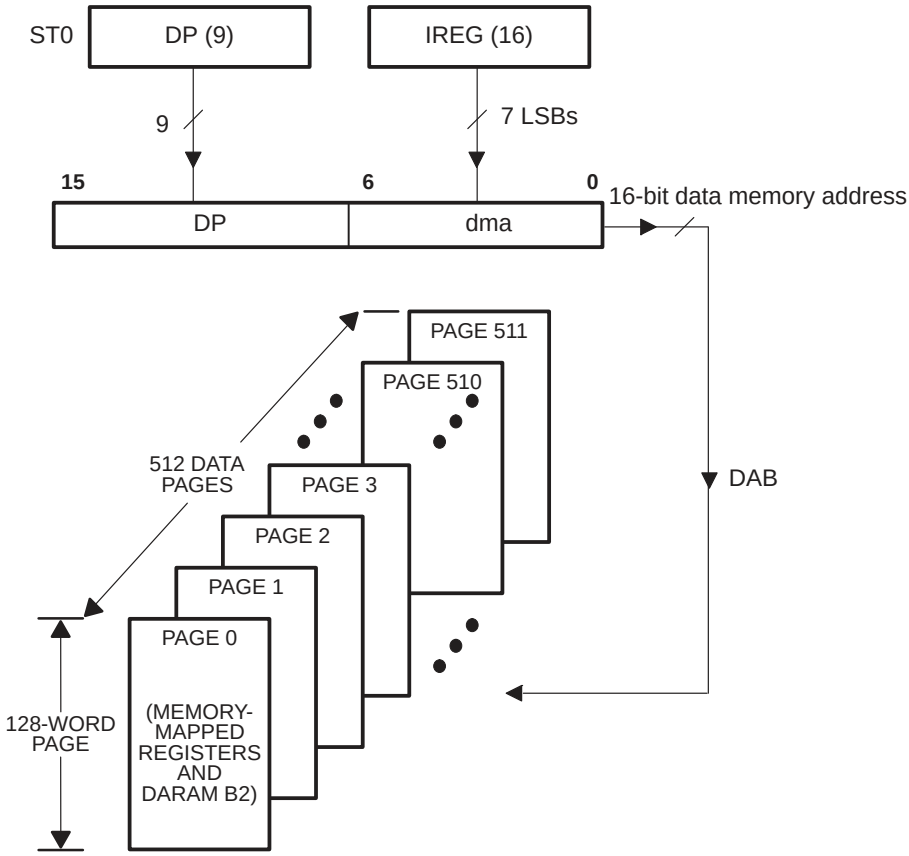
Topic **Page**

5.1 Direct Addressing

In the direct memory addressing mode, the instruction contains the lower 7 bits of the data memory address (dma). The 7-bit dma is concatenated with the 9 bits of the data memory page pointer (DP) in status register 0 to form the full 16-bit data memory address. This 16-bit data memory address is placed on an internal direct data memory address bus (DAB). The DP points to one of 512 possible data memory pages and the 7-bit address in the instruction points to one of 128 words within that data memory page. You can load the DP bits by using the LDP or the LST #0 instruction.

Figure 5–1 illustrates how the 16-bit data memory address is formed.

Figure 5–1. Direct Addressing

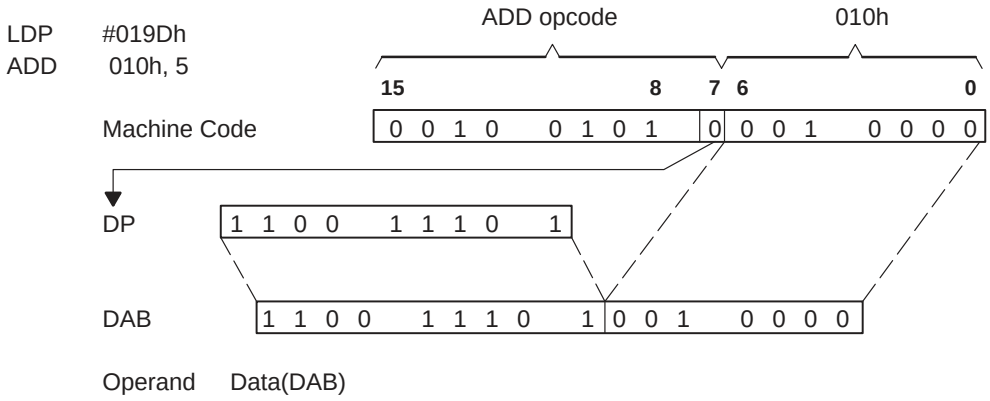


Note:

The DP is not initialized by reset and, therefore, is undefined after power-up. The ’C5x development tools, however, use default values for many parameters, including the DP. Because of this, programs that do not explicitly initialize the DP may execute improperly, depending on whether they are executed on a ’C5x device or with a development tool. Thus, it is critical that all programs initialize the DP in software.

Figure 5–2 illustrates the direct addressing mode. Bits 15 through 8 contain the opcode. Bit 7, with a value of 0, defines the addressing mode as direct, and bits 6 through 0 contain the dma.

Figure 5–2. Direct Addressing Mode

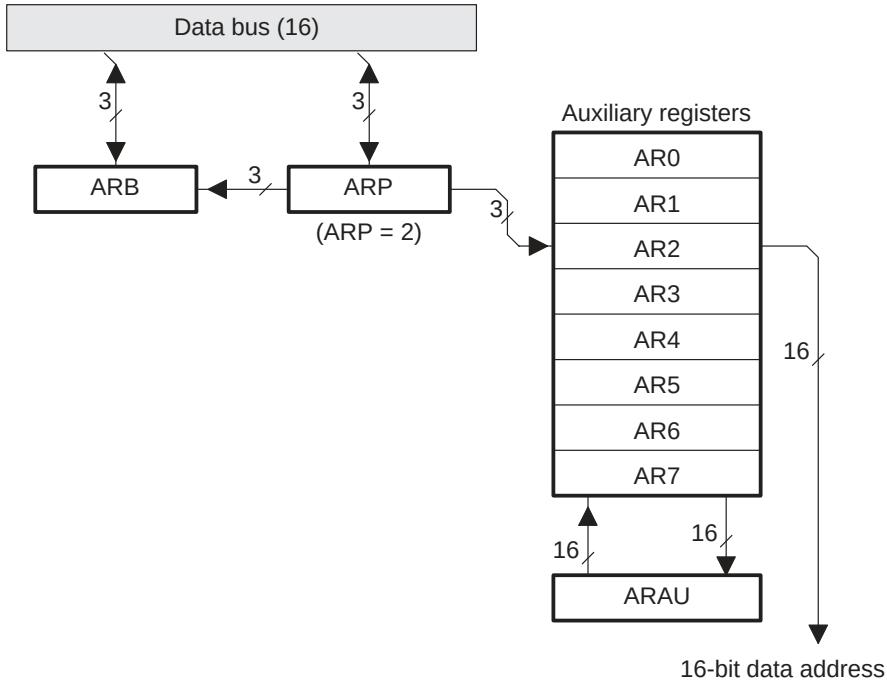


Note: DAB is the 16-bit internal data memory address bus.

5.2 Indirect Addressing

Eight 16-bit auxiliary registers (AR0–AR7) provide flexible and powerful indirect addressing. In indirect addressing, any location in the 64K-word data memory space can be accessed using a 16-bit address contained in an AR. Figure 5–3 shows the hardware for indirect addressing.

Figure 5–3. Indirect Addressing



To select a specific AR, load the auxiliary register pointer (ARP) with a value from 0 through 7, designating AR0 through AR7, respectively. The register pointed to by the ARP is referred to as the *current auxiliary register* (current AR). You can load the address into the AR using the LAR instruction and you can change the content of the AR by the:

- ADRK instruction
- MAR instruction
- SBRK instruction
- Indirect addressing field of any instruction supporting indirect addressing.

The content of the current AR is used as the address of the data memory operand. After the instruction uses the data value, the content of the current AR can be incremented or decremented by the auxiliary register arithmetic unit (ARAU), which implements unsigned 16-bit arithmetic.

The ARAU performs auxiliary register arithmetic operations in the decode phase of the pipeline (when the instruction specifying the operation is being decoded). This allows the address to be generated before the decode phase of the next instruction. The content of the current AR is incremented or decremented after it is used in the current instruction.

You can load the ARs via the data bus by using memory-mapped writes to the ARs. The following instructions can write to the memory-mapped ARs:

APL	OPL	SAMM	XPL
BLDD	SACH	SMMR	
LMMR	SACL	SPLK	

Be careful when using these memory-mapped loads of the ARs because, in this case, the memory-mapped ARs are modified in the execute phase of the pipeline. This causes a pipeline conflict if one of the next two instruction words modifies that AR. For further information on the pipeline and possible pipeline conflicts, see Chapter 7, *Pipeline*.

There are two ways to use the ARs for purposes other than referencing data memory addresses:

- Use the ARs to support conditional branches, calls, and returns by using the CMPR instruction. This instruction compares the content of the current AR with the content of the auxiliary register compare register (ARCR) and puts the result in the test/control (TC) flag bit of status register ST1.
- Use the ARs for temporary storage by using the LAR instruction to load a value into the AR and the SAR instruction to store the AR value to a data memory location.

5.2.1 Indirect Addressing Options

The ’C5x provides four indirect addressing options:

- **No increment or decrement.** The instruction uses the content of the current AR as the data memory address, but neither increments nor decrements the content of the current AR.
- **Increment or decrement by one.** The instruction uses the content of the current AR as the data memory address and then increments or decrements the content of the current AR by 1.
- **Increment or decrement by an index amount.** The value in INDX is the index amount. The instruction uses the content of the current AR as the data memory address and then increments or decrements the content of the current AR by the index amount.

- **Increment or decrement by an index amount using reverse carry.** The value in INDX is the index amount. The instruction uses the content of the current AR as the data memory address and then increments or decrements the content of the current AR by the index amount. The addition or subtraction is done using reverse carry propagation.

The contents of the current AR are used as the address of the data memory operand. Then, the ARAU performs the specified mathematical operation on the indicated AR. Additionally, the ARP can be loaded with a new value. All indexing operations are performed on the current AR in the same cycle as the original instruction decode phase of the pipeline.

Indirect auxiliary register addressing lets you make post-access adjustments of the current AR. The adjustment may be an increment or decrement by one or may be based upon the contents of the INDX. To maintain compatibility with the 'C2x devices, clear the NDX bit in the PMST. In the 'C2x architecture, the current AR can be incremented or decremented by the value in the AR0. When the NDX bit is cleared, every AR0 modification or LAR write also writes the ARCR and INDX with the same value. Subsequent modifications of the current ARs with indexed addressing will use the INDX, therefore maintaining compatibility with existing 'C2x code. The NDX bit is cleared at reset.

The bit-reversed addressing modes (see subsection 5.2.3 on page 5-12) helps you achieve efficient I/O by the resequencing of data points in a radix-2 fast Fourier transform (FFT) program. The direction of carry propagation in the ARAU is reversed when bit-reversed addressing is selected, and INDX is added to/subtracted from the current AR. Normally, this addressing mode requires that INDX first be set to a value corresponding to one-half of the array's size, and that the current AR be set to the base address of the data (the first data point).

The following indirect-addressing symbols are used in the 'C5x assembly language instructions:

***** **No increment or decrement.** Content of the current AR is used as the data memory address and is neither incremented nor decremented.

***+** **Increment by 1.** Content of the current AR is used as the data memory address. After the memory access, the content of the current AR is incremented by 1.

***–** **Decrement by 1.** Content of current AR is used as the data memory address. After the memory access, the content of the current AR is decremented by 1.

***0+** **Increment by index amount.** Content of current AR is used as the data memory address. After the memory access, the content of INDX is added to the content of the current AR.

***0–** **Decrement by index amount.** Content of current AR is used as the data memory address. After the memory access, the content of INDX is subtracted from the content of the current AR.

***BR0+** **Increment by index amount, adding with reverse carry.** Content of current AR is used as the data memory address. After the memory access, the content of INDX with reverse carry propagation is added to the content of the current AR.

***BR0–** **Decrement by index amount, subtracting with reverse carry.** Content of current AR is used as the data memory address. After the memory access, the content of INDX with reverse carry propagation is subtracted from the content of the current AR.

5.2.2 Indirect Addressing Opcode Format

Indirect addressing can be used with all instructions except those with immediate operands or with no operands. The indirect addressing format is shown in Figure 5–4 and described in Table 5–1.

Table 5–3 on page 5-9 shows the instruction field bit values, notation, and operation used for indirect addressing. Example 5–1 through Example 5–8 illustrate the indirect addressing formats. Example 5–9 shows an indirect addressing routine.

Figure 5–4. Indirect Addressing Opcode Format Diagram

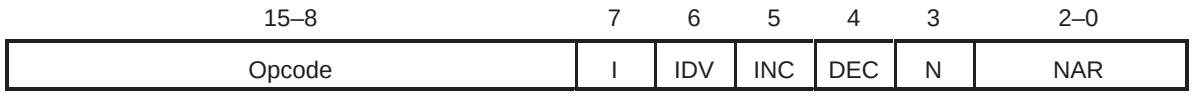


Table 5–1. Indirect Addressing Opcode Format Summary

Bit	Name	Description
15–8		Opcode. This 8-bit field is the opcode for the instruction.
7	I	Addressing mode bit. This 1-bit field determines the addressing mode.
		I = 0 Direct addressing mode.
		I = 1 Indirect addressing mode.

Table 5–1. Indirect Addressing Opcode Format Summary (Continued)

Bit	Name	Description
6	IDV	Index register bit. This 1-bit field determines whether the INDX is used to increment or decrement the current AR. The IDV bit works in conjunction with the INC and DEC bits to determine the arithmetic operation. IDV = 0 The INDX is not used in the arithmetic operation. An increment or decrement (if any) by 1 occurs to the current AR. IDV = 1 The INDX is used in the arithmetic operation. An increment or decrement (if any) by the contents of INDX or by reverse carry propagation occurs to the current AR.
5	INC	Auxiliary register increment bit. This 1-bit field determines whether the current AR is incremented. The INC bit works in conjunction with the IDV and DEC bits to determine the arithmetic operation. INC = 0 The current AR is not incremented. INC = 1 The current AR is incremented as determined by the IDV bit.
4	DEC	Auxiliary register decrement bit. This 1-bit field determines whether the current AR is decremented. The DEC bit works in conjunction with the IDV and INC bits to determine the arithmetic operation. See Table 5–2 for specific arithmetic operations. DEC = 0 The current AR is not decremented. DEC = 1 The current AR is decremented as determined by the IDV bit.
3	N	Next auxiliary register indicator bit. This 1-bit field determines whether the instruction will change the ARP value. N = 0 The content of the ARP will remain unchanged. N = 1 The content of NAR will be loaded into the ARP, and the old ARP value is loaded into the auxiliary register buffer (ARB) of status register ST1.
2–0	NAR	Next auxiliary register value bits. This 3-bit field contains the value of the next auxiliary register. If the N bit is set, NAR is loaded into the ARP.

Table 5–2. Indirect Addressing Arithmetic Operations

Bit values			
IDV	**INC**	**DEC**	**Arithmetic Operation Performed on Current AR**
0	0	0	No operation on current AR
0	0	1	(Current AR) – 1 → current AR
0	1	0	(Current AR) + 1 → current AR
0	1	1	Reserved
1	0	0	(Current AR) – INDX [reverse carry propagation] → current AR
1	0	1	(Current AR) – INDX → current AR
1	1	0	(Current AR) + INDX → current AR
1	1	1	(Current AR) + INDX [reverse carry propagation] → current AR

Table 5–3. Instruction Field Bit Values for Indirect Addressing

Instruction Field Bit Values								
15–8	**7**	**6**	**5**	**4**	**3**	**2–0**	**Notation**	**Operation**
← Opcode →	1	0	0	0	0	← NAR →	*****	No operation on current AR
← Opcode →	1	0	0	0	1	← NAR →	***, AR***n*	NAR → ARP
← Opcode →	1	0	0	1	0	← NAR →	***–**	(Current AR) – 1 → current AR
← Opcode →	1	0	0	1	1	← NAR →	***–, AR***n*	(Current AR) – 1 → current AR, NAR → ARP
← Opcode →	1	0	1	0	0	← NAR →	***+**	(Current AR) + 1 → current AR
← Opcode →	1	0	1	0	1	← NAR →	***+, AR***n*	(Current AR) + 1 → current AR, NAR → ARP
← Opcode →	1	1	0	0	0	← NAR →	***BR0–**	(Current AR) – *rc*INDX → current AR
← Opcode →	1	1	0	0	1	← NAR →	***BR0–, AR***n*	(Current AR) – *rc*INDX → current AR, NAR → ARP
← Opcode →	1	1	0	1	0	← NAR →	***0–**	(Current AR) – INDX → current AR
← Opcode →	1	1	0	1	1	← NAR →	***0–, AR***n*	(Current AR) – INDX → current AR, NAR → ARP
← Opcode →	1	1	1	0	0	← NAR →	***0+**	(Current AR) + INDX → current AR
← Opcode →	1	1	1	0	1	← NAR →	***0+, AR***n*	(Current AR) + INDX → current AR, NAR → ARP

Table 5–3. Instruction Field Bit Values for Indirect Addressing (Continued)

Instruction Field Bit Values								
15–8	7	6	5	4	3	2–0	Notation	Operation
← Opcode →	1	1	1	1	0	← NAR →	***BR0+**	(Current AR) + *rc*INDX → current AR
← Opcode →	1	1	1	1	1	← NAR →	***BR0+, AR**n*	(Current AR) + *rc*INDX → current AR, NAR → ARP

Example 5–1. Indirect Addressing With No Change to AR

```
ADD *,8
```

0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0

In Example 5–1, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The current AR is not changed. The instruction word is 2880h.

Example 5–2. Indirect Addressing With Autodecrement

```
ADD *-,8
```

0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	0

In Example 5–2, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The current AR is decremented by 1. The instruction word is 2890h.

Example 5–3. Indirect Addressing With Autoincrement

```
ADD *+,8
```

0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0

In Example 5–3, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The current AR is incremented by 1. The instruction word is 28A0h.

Example 5–4. Indirect Addressing With Autoincrement and Change AR

```
ADD *+,8,AR3
```

0	0	1	0	1	0	0	0	1	0	1	0	1	0	1	1

In Example 5–4, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The current AR is incremented by 1. The auxiliary register pointer (ARP) is loaded with the value 3 for subsequent instructions. The instruction word is 28ABh.

Example 5–5. Indirect Addressing With INDX Subtracted from AR

```
ADD *0–,8
```

0	0	1	0	1	0	0	0	1	1	0	1	0	0	0	0

In Example 5–5, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The content of INDX is subtracted from the current AR. The instruction word is 28D0h.

Example 5–6. Indirect Addressing With INDX Added to AR

```
ADD *0+,8
```

0	0	1	0	1	0	0	0	1	1	1	0	0	0	0	0

In Example 5–6, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The content of INDX is added to the current AR. The instruction word is 28E0h.

Example 5–7. Indirect Addressing With INDX Subtracted from AR With Reverse Carry

```
ADD *BR0–,8
```

0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0

In Example 5–7, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The content of INDX with reverse carry propagation is subtracted from the current AR. The instruction word is 28C0h.

Example 5–8. Indirect Addressing With INDX Added to AR With Reverse Carry

```
ADD *BR0+,8
```

0	0	1	0	1	0	0	0	1	1	1	1	0	0	0	0

In Example 5–8, the content of the data memory address, defined by the content of the current AR, is shifted left 8 bits and added to the ACC. The content of INDX with reverse carry propagation is added to the current AR. The instruction word is 28F0h.

Example 5–9. Indirect Addressing Routine

```
* This routine uses indirect addressing to calculate the following equation:
*
*                   10
*                  -----
*                  \     X(I) x Y(I)
*                   /
*                  -----
*                  I = 1
*
* The routine assumes that the X values are located in on-chip RAM block B0,
* and the Y values in block B1. The efficiency of the routine is due to the
* use of indirect addressing and the repeat instruction.
*
SERIES MAR    *,AR4         ;ARP POINTS TO ADDRESS REGISTER 4.
       SETC   CNF           ;CONFIGURE BLOCK B0 AS PROGRAM MEMORY.
       LAR    AR4,#0300h    ;POINT AT BEGINNING OF DATA MEMORY.
       RPTZ   #9            ;CLEAR ACC AND PREG; REPEAT NEXT INST. 10 TIMES
       MAC    0FF00h,*+     ;MULTIPLY AND ACCUMULATE; INCREMENT AR4.
              APAC          ;ACCUMULATE LAST PRODUCT.
              RET           ;ACCUMULATOR CONTAINS RESULT.
```

5.2.3 Bit-Reversed Addressing

In the bit-reversed addressing mode, INDX specifies one-half the size of the FFT. The value contained in the current AR must be equal to 2^{n-1}, where n is an integer, and the FFT size is 2^n. An auxiliary register points to the physical location of a data value. When you add INDX to the current AR using bit-reversed addressing, addresses are generated in a bit-reversed fashion.

Assume that the auxiliary registers are eight bits long, that AR2 represents the base address of the data in memory ($0110\ 0000_2$), and that INDX contains the value $0000\ 1000_2$. Example 5–10 shows a sequence of modifications to AR2 and the resulting values of AR2. Table 5–4 shows the relationship of the bit pattern of the index steps and the four LSBs of AR2, which contain the bit-reversed address.

Example 5–10. Sequence of Auxiliary Register Modifications in Bit-Reversed Addressing

```
*BR0+  ;AR2   =   0110 0000    (0th value)
*BR0+  ,AR2   =   0110 1000    (1st value)
*BR0+  ;AR2   =   0110 0100    (2nd value)
*BR0+  ;AR2   =   0110 1100    (3rd value)
*BR0+  ;AR2   =   0110 0010    (4th value)
*BR0+  ;AR2   =   0110 1010    (5th value)
*BR0+  ;AR2   =   0110 0110    (6th value)
*BR0+  ;AR2   =   0110 1110    (7th value)
```

Table 5–4. Bit-Reversed Addresses

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

5.3 Immediate Addressing

In immediate addressing, the instruction word(s) contains the value of the immediate operand. The ’C5x has both 1-word (8-bit, 9-bit, and 13-bit constant) short immediate instructions and 2-word (16-bit constant) long immediate instructions. Table 5–5 lists the instructions that support immediate addressing.

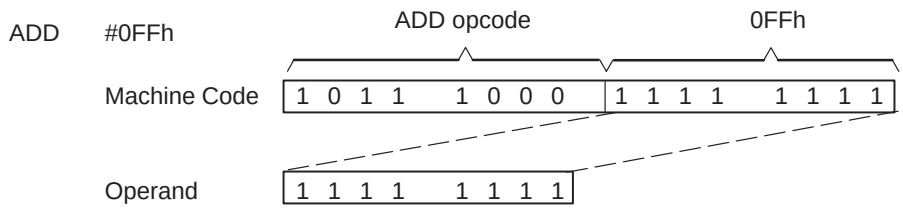
Table 5–5. Instructions That Support Immediate Addressing

Short Immediate (1-Word)			Long Immediate (2-Word)	
8-Bit Constant	**9-Bit Constant**	**13-Bit Constant**	**16-Bit Constant**	
ADD	LDP	MPY	ADD	OR
ADRK			AND	RPT
LACL			APL	RPTZ
LAR			CPL	SPLK
RPT			LACC	SUB
SBRK			LAR	XOR
SUB			MPY	XPL
			OPL	

5.3.1 Short Immediate Addressing

In short immediate instructions, the operand is contained within the instruction machine code. Figure 5–5 shows an example of the short immediate mode. Note that in this example, the lower 8 bits are the operand and will be added to the ACC by the CALU.

Figure 5–5. Short Immediate Addressing Mode



5.3.2 Long Immediate Addressing

In long immediate instructions, the operand is contained in the second word of a two-word instruction. There are two long immediate addressing modes:

- One-operand instructions
- Two-operand instructions

5.3.2.1 Long Immediate Addressing with Single/No Data Memory Access

Figure 5–6 shows an example of long immediate addressing with no data memory access. In Figure 5–6, the second word of the 2-word instruction is added to the ACC by the CALU.

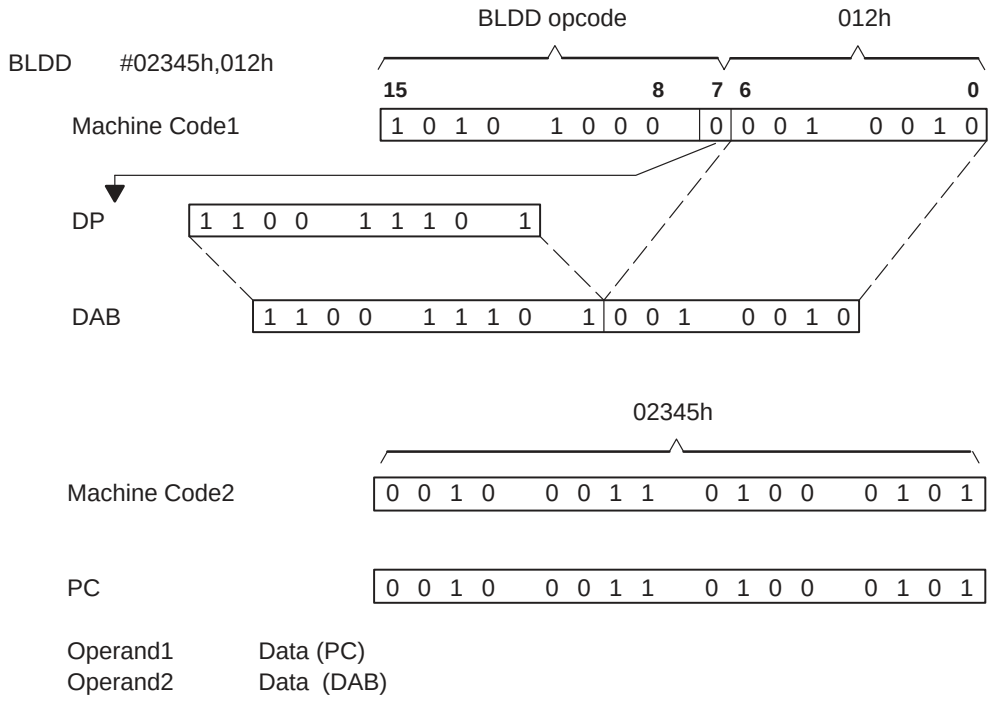
Figure 5–6. Long Immediate Addressing Mode — No Data Memory Access

5.3.2.2 Long Immediate Addressing with Dual Data Memory Access

The long immediate addressing also could apply for a second data memory access for the execution of the instruction. The prefetch counter (PFC) is pushed onto the microcall stack (MCS), and the long immediate value is loaded into the PFC. The program address/data bus is then used for the operand fetch or write. At the completion of the instruction, the MCS is popped back to the PFC, the program counter (PC) is incremented by two, and execution continues. The PFC is used so that when the instruction is repeated, the address generated can be autoincremented.

Figure 5–7 shows an example of long immediate addressing with two operands. In Figure 5–7, the source address (OPERAND1) is fetched via PAB, and the destination address (OPERAND2) uses the direct addressing mode. Bits 15 through 8 of machine code1 contain the opcode. Bit 7, with a value of 0, defines the addressing mode as direct, and bits 6 through 0 contain the dma.

Figure 5–7. Long Immediate Addressing Mode — Two Operands



Note: DAB is the 16-bit internal data memory address bus.

5.4 Dedicated-Register Addressing

The dedicated-registered addressing mode operates like the long immediate addressing mode, except that the address comes from one of two special-purpose memory-mapped registers in the CPU: the block move address register (BMAR) and the dynamic bit manipulation register (DBMR). The advantage of this addressing mode is that the address of the block of memory to be acted upon can be changed during execution of the program. The syntax for dedicated-register addressing can be stated in one of two ways:

- Specify BMAR by its predefined symbol:

  ```
  BLDD BMAR,DAT100 ;DP = 0. BMAR contains the value 200h.
  ```

  The content of data memory location 200h is copied to data memory location 100 on the current data page.

- Exclude the immediate value from a parallel logic unit (PLU) instruction:

  ```
  OPL    DAT10  ;DP = 6. DBMR contains the value FFF0h.
                ;Address 030Ah contains the value 01h
  ```

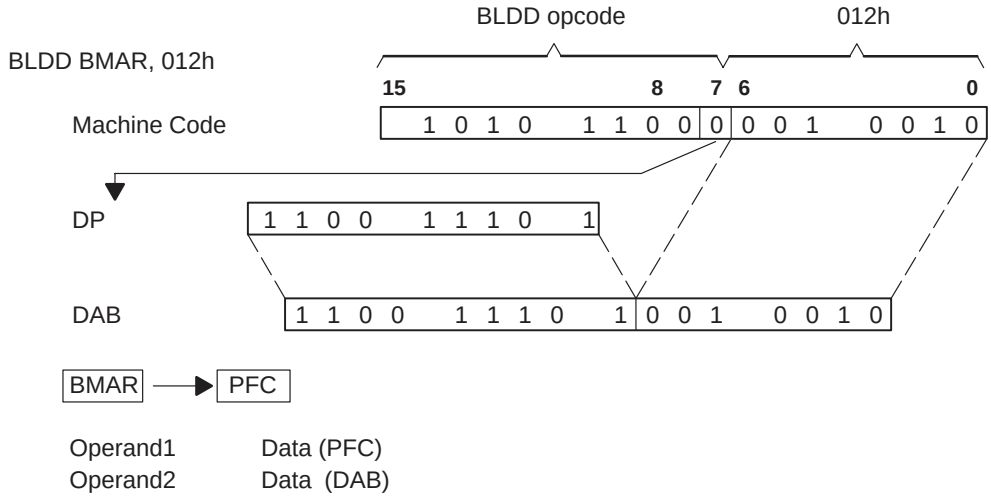
  The content of data memory location 030Ah is ORed with the content of the DBMR. The resulting value FFF1h is stored back in memory location 030Ah.

5.4.1 Using the Contents of the BMAR

The BLDD, BLDP, and BLPD instructions use the BMAR to point at the source or destination space of a block move. The MADD and MADS instructions also use the BMAR to address an operand in program memory for a multiply-accumulate operation.

Figure 5–8 shows how the BMAR is used in the dedicated-register addressing mode. Bits 15 through 8 of the machine code contain the opcode. Bit 7, with a value of 0, defines the addressing mode as direct, and bits 6 through 0 contain the dma.

Figure 5–8. Dedicated-Register Addressing Using the BMAR



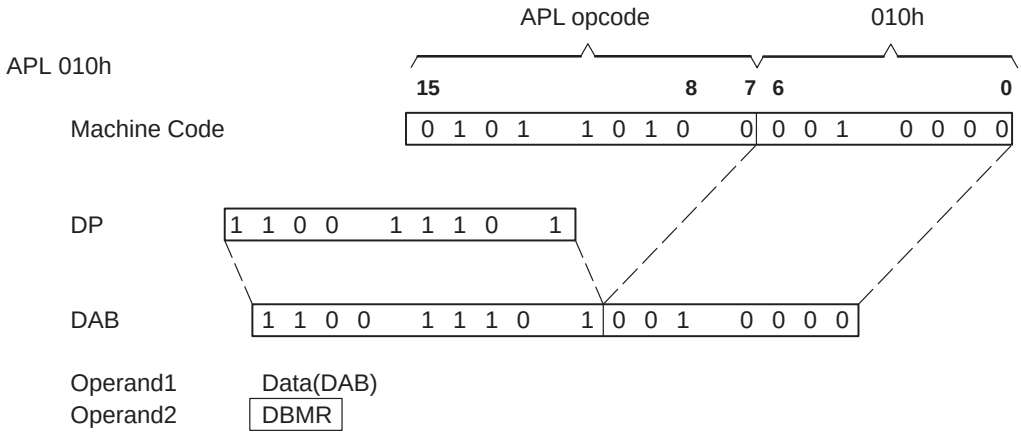
Note: DAB is the 16-bit internal data memory address bus.

5.4.2 Using the Contents of the DBMR

The APL, CPL, OPL, and XPL instructions use the PLU and the contents of the DBMR when an immediate value is not specified as one of the operands.

Figure 5–9 illustrates how the DBMR is used as an AND mask in the APL instruction. Bits 15 through 8 of the machine code contain the opcode. Bit 7, with a value of 0, defines the addressing mode as direct, and bits 6 through 0 contain the dma.

Figure 5–9. Dedicated-Register Addressing Using the DBMR



Note: DAB is the 16-bit internal data memory address bus.

5.5 Memory-Mapped Register Addressing

With memory-mapped register addressing, you can modify the memory-mapped registers without affecting the current data page pointer value. In addition, you can modify any scratch pad RAM (DARAM B2) location or data page 0. The memory-mapped register addressing mode operates like the direct addressing mode, except that the 9 MSBs of the address are forced to 0 instead of being loaded with the contents of the DP. This allows you to address the memory-mapped registers of data page 0 directly without the overhead of changing the DP or auxiliary register.

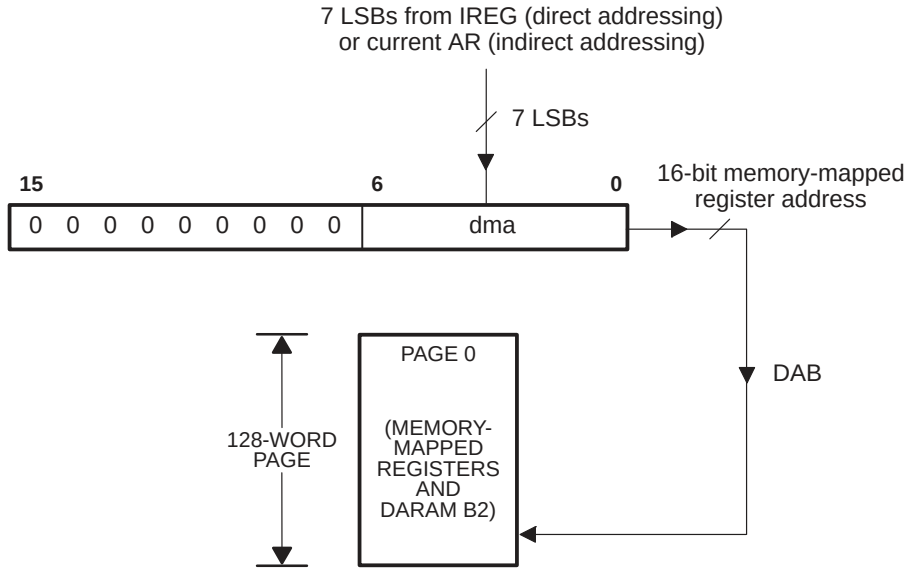
The following instructions operate in the memory-mapped register addressing mode. Using these instructions does not affect the contents of the DP:

- LAMM — Load accumulator with memory-mapped register
- LMMR — Load memory-mapped register
- SAMM — Store accumulator in memory-mapped register
- SMMR — Store memory-mapped register

Figure 5–10 illustrates how this is done by forcing the 9 MSBs of the data memory address to 0, regardless of the current value of the DP when direct addressing is used or of the current AR value when indirect addressing is used.

Example 5–11 uses memory-mapped register addressing in the direct addressing mode and Example 5–12 uses the indirect addressing mode.

Figure 5–10. Memory-Mapped Register Addressing



Example 5–11. Memory-Mapped Register Addressing in the Indirect Addressing Mode

```
SAMM   *+          ;STORE ACC TO PMST REGISTER
```

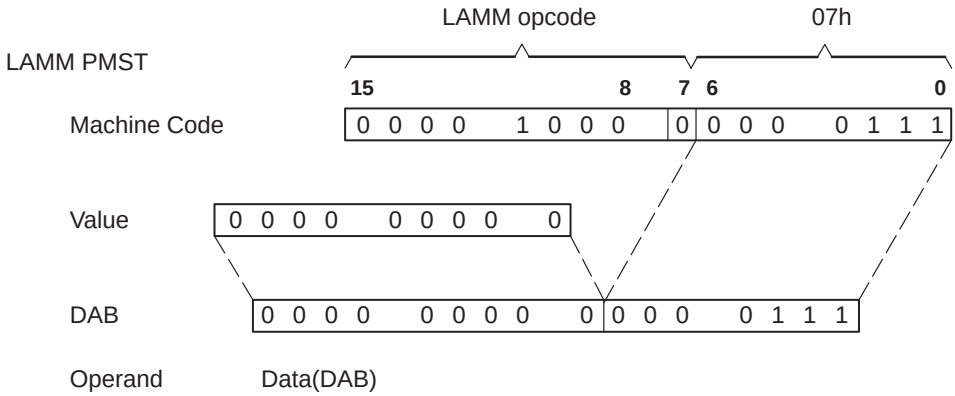
In Example 5–11, assume that ARP = 3 and AR3 = FF07h. The content of the ACC is stored to the PMST (address 07h) pointed at by the 7 LSBs of AR3.

Example 5–12. Memory-Mapped Register Addressing in the Direct Addressing Mode

```
LAMM   07h         ;ACC = PMST
```

In Example 5–12, assume that DP = 0184h and TEMP1 = 8060h. The content of memory location 07h (PMST) is loaded into the ACC. Figure 5–11 illustrates memory-mapped register addressing in the direct addressing mode.

Figure 5–11. Memory-Mapped Addressing in the Direct Addressing Mode



Note: DAB is the 16-bit internal data memory address bus.

5.6 Circular Addressing

Many algorithms such as convolution, correlation, and finite impulse response (FIR) filters can use circular buffers in memory to implement a sliding window, which contains the most recent data to be processed. The ’C5x supports two concurrent circular buffers operating via the ARs. The following five memory-mapped registers control the circular buffer operation:

- CBSR1 — Circular buffer 1 start register
- CBSR2 — Circular buffer 2 start register
- CBER1 — Circular buffer 1 end register
- CBER2 — Circular buffer 2 end register
- CBCR — Circular buffer control register

The 8-bit CBCR enables and disables the circular buffer operation and is defined in subsection 4.4.1, *Circular Buffer Control Register (CBCR)*, on page 4-6.

To define circular buffers, you first load the start and end addresses into the corresponding buffer registers; next, load a value between the start and end registers for the circular buffer into an AR. Load the proper AR value, and set the corresponding circular buffer enable bit in the CBCR. Note that you must not enable the same AR for both circular buffers; if you do, unexpected results occur. The algorithm for circular buffer addressing below shows that the test of the AR value is performed before any modifications:

If (ARn = CBER) and (any AR modification),
Then: ARn = CBSR.
Else: ARn = ARn + step.

If ARn = CBER and no AR modification occurs, the current AR is not modified and is still equal to CBER. When the current AR = CBER, any AR modification (increment or decrement) will set the current AR = CBSR. Example 5–13 illustrates the operation of circular addressing.

Example 5–13. Circular Addressing

```
mar    *,ar6
lpd    #,0

splk   #200h,CBSR1   ;  Circular buffer start register
splk   #203h,CBER1   ;  Circular buffer end register
splk   #0Eh,CBCR     ;  Enable AR6 pointing to buffer 1

lar    ar6,#200h     ;  Case 1
lacc   *             ;  AR6 = 200h

lar    ar6,#203h     ;  Case 2
lacc   *             ;  AR6 = 203h

lar    ar6,#200h     ;  Case 3
lacc   *+            ;  AR6 = 201h

lar    ar6,#203h     ;  Case 4
lacc   *+            ;  AR6 = 200h

lar    ar6,#200h     ;  Case 5
lacc   *–            ;  AR6 = 1FFh

lar    ar6,#203h     ;  Case 6
lacc   *–            ;  AR6 = 200h

lar    ar6,#202h     ;  Case 7
adrk   2             ;  AR6 = 204h

lar    ar6,#203h     ;  Case 8
adrk   2             ;  AR6 = 200h
```

In circular addressing, the step is the quantity that is being added to or subtracted from the specified AR. Take care when using a step of greater than 1 to modify the AR pointing to an element of the circular buffer. If an update to an AR generates an address outside the range of the circular buffer, the ARAU does not detect this situation, and the buffer does not wrap around. AR updates are performed as described in Section 5.2, *Indirect Addressing*. Because of the pipeline, there is a two-cycle latency between configuring the CBCR and performing AR modifications.

Circular buffers can be used in increment- or decrement-type updates. For incrementing the value in the AR, the value in CBER must be greater than the value in CBSR. For decrementing the value in the AR, the value in CBSR must be greater than the value in CBER.

Chapter 6

Assembly Language Instructions

The ’C5x instruction set supports numerically intensive signal-processing operations as well as general-purpose applications, such as multiprocessing and high-speed control. The instruction set is a superset of the ’C1x and ’C2x instruction sets and is source-code upward compatible with both devices.

Section 6.3, *Instruction Set Descriptions*, describes individual instructions in detail. Chapter 5, *Addressing Modes*, discusses the addressing modes associated with the instruction set. Section C.4, *’C2x-to-’C5x Instruction Set Mapping*, includes a table that maps ’C2x instructions to ’C5x instructions. Note that the Texas Instruments ’C5x assembler accepts ’C2x instructions as well as ’C5x instructions.

Topic **Page**

6.1 Instruction Set Symbols and Notations

For the sake of convenience and as a memory aid, this chapter uses many symbols and notations while describing the assembly language instructions. This section provides a centralized list of definitions for these symbols and notations.

6.1.1 Symbols and Abbreviations Used in the Instruction Set Opcodes

Table 6–1 explains the symbols and abbreviations used in the opcode of the instruction set summaries (Table 6–4 through Table 6–10) and instruction set descriptions (Section 6.3, page 6-22).

Table 6–1. Instruction Set Opcode Symbols and Abbreviations

Symbol	Meaning
AAA AAAA	The data memory address bits. When indirect addressing (I = 1) is being used, the seven As are the seven least significant bits (LSBs) of a data memory address. For indirect addressing, the seven As are bits that control auxiliary register manipulation (see Section 5.2, *Indirect Addressing*, on page 5-4.)
ARX	A 3-bit value used in the LAR and SAR instructions to designate which auxiliary register (0–7) will be loaded (LAR) or have its contents stored (SAR).
BITX	A 4-bit value (called the bit code) that determines which bit of a designated data memory value will be tested by the BIT instruction.
CM	A 2-bit value that determines the comparison performed by the CMPR instruction.
I	The addressing mode bit. When I = 0, the direct addressing mode is being used. When I =1, the indirect addressing mode is being used.
kkkk kkkk	An 8-bit constant used in short immediate addressing for the ADD, ADRK, LACL, LAR, RPT, SBRK, and SUB instructions.
k kkkk kkkk	A 9-bit constant used in short immediate addressing for the LDP instruction.
k kkkk kkkk kkkk	A 13-bit constant used in short immediate addressing for the MPY instruction.
I NTR #	The interrupt vector number. A 5-bit value representing a number from 0 to 31. The INTR instruction uses this number to change program control to one of the 32 interrupt vector addresses.
PM	A 2-bit value copied into the product shift mode (PM) bits of status register ST1 by the SPM instruction.
SHF	A 3-bit shift value for the SACH and SACL instructions.
SHFT	A 4-bit shift value for the ADD, AND, BSAR, LACC, OR, SUB, and XOR instructions.

Table 6–1. Instruction Set Opcode Symbols and Abbreviations (Continued)

Symbol	Meaning
N	A 1-bit field for the XC instruction indicating the number of instructions (one or two) to conditionally execute. If N = 0, one instruction will execute. If N = 1, two instructions will execute.
TP	A 2-bit value used by the conditional execution instructions to represent the following conditions: **TP** — **Condition** 0 0 — $\overline{BIO}$ pin low 0 1 — TC = 1 1 0 — TC = 0 (NTC) 1 1 — None of the above conditions
ZLVC ZLVC	Two 4-bit fields designating the following bit conditions to be tested and the bit states: **Bit** — **Condition** Z — ACC = 0 L — ACC < 0 V — Overflow C — Carry A conditional instruction contains two of these 4-bit fields. The 4-LSB field of the instruction is a mask field. A 1 in a mask bit indicates that the corresponding condition is being tested. The second 4-bit field (bits 4–7) indicates the state of the conditions being tested. For example, to test for ACC ≥ 0, the Z and L bits of the 4-LSB field are set, while the V and C bits are not set. When the Z bit is set, it indicates to test for the condition ACC = 0; when the L bit is set, it indicates to test for the condition ACC ≥ 0. The conditions possible with these 8 bits are shown in the BCND, BCNDD, CC, CCD, RETC, RETCD, and XC instructions. To determine if the conditions are met, the 4-LSB field is ANDed with the 4-bit field containing the state of the conditions. If any bits are set, the conditions are met.
+ 1 word	The second word of a two-word opcode. This second word contains a 16-bit constant. Depending on the instruction, this constant is a long immediate value, a program memory address, or an address for an I/O port or an I/O-mapped register.

6.1.2 Symbols and Abbreviations Used in the Instruction Set Descriptions

Table 6–2 explains the symbols and abbreviations used in the instruction set descriptions (Section 6.3, page 6-22).

Table 6–2. Instruction Set Descriptions Symbols and Abbreviations

Symbol	Meaning
ACC	Accumulator
ACCB	Accumulator buffer
ACCH	Accumulator high byte, ACC(31–16)
ACCL	Accumulator low byte, ACC(15–0)
addr	16-bit data memory address
ALU	Arithmetic logic unit
AR	Auxiliary register
ARB	Auxiliary register buffer (in ST1). This register stores the previous ARP value.
ARCR	Auxiliary register compare register
ARn	A value n from 0 to 7 designating the next auxiliary register (AR), the register that will be pointed to by the ARP when the instruction is complete
ARP	Auxiliary register pointer (in ST0). This register points to the current auxiliary register (AR).
AVIS	Address visibility bit (in PMST)
BIO	Branch control input
bit code	A 4-bit value that determines which bit of a designated data memory value will be tested by the BIT instruction.
BMAR	Block move address register
BRAF	Block repeat active flag bit (in PMST)
C	Carry bit (in ST1)
CNF	On-chip RAM configuration control bit (in ST1)
cond	An operand representing a condition used by instructions that execute conditionally.
current AR	The current auxiliary register; that is, the auxiliary register (AR) pointed to by the ARP.
D	Data memory address field
dst	Destination address field
DATn	Label assigned to data memory location n
DBMR	Dynamic bit manipulation register
dma	The 7 LSBs of a data memory address.

Table 6–2. Instruction Set Descriptions Symbols and Abbreviations (Continued)

Symbol	Meaning
DP	Data memory page pointer bits (in ST0)
HM	Hold mode bit (in ST1)
ind	Indirect addressing operand (see Section 5.2, *Indirect Addressing*, on page 5-4.)
INTM	Interrupt mode flag bit (in ST0)
k	Short immediate operand (an 8-, 9-, or 13-bit constant)
K	A value from 0 to 31 indicating one of the 32 interrupt vector locations. The INTR instruction forces a branch to the location referenced by K.
lk	Long immediate operand (a 16-bit constant)
MCS	Microcall stack
MP/$\overline{\text{MC}}$	Microprocessor/Microcomputer bit (in PMST)
n	A value of 1 or 2 designating the number of words following the XC instruction.
OV	Overflow bit (in ST0)
OVLY	RAM overlay bit (in PMST)
OVM	Overflow mode bit (in ST0)
NDX	Enable extra index register bit (in PMST)
PA	A 16-bit address for an I/O port or an I/O-mapped register ($0 \leq PA \leq 65535$)
PAER	Block Repeat Program Address End Register
PASR	Block Repeat Program Address Start Register
PC	Program counter
PFC	Prefetch counter
PGMn	Label assigned to program memory location n
PM	Product shift mode bits (in ST1)
pma	A 16-bit program memory address
PREG	Product register
RAM bit	Program RAM enable bit (in PMST)
RPTC	Repeat counter
shift	A 4-bit shift value from 0–15
shift2	A 3-bit shift value from 0–7
src	Source address field

Table 6–2. Instruction Set Descriptions Symbols and Abbreviations (Continued)

Symbol	Meaning
STm	Status register m (m = 0 or 1)
SXM	Sign-extension mode bit (in ST1)
TREGn	Temporary register n (n = 0, 1, or 2)
TC	Test/control bit (in ST1)
TOS	Top of stack
TRM	Enable multiple TREGs bit (in PMST)
x	A value from 0 to 7 designating one of the eight auxiliary registers (AR0–AR7).
XF	XF pin status bit (in ST1)

6.1.3 Notations Used in the Instruction Set Descriptions

Special notations have been used to describe the execution of the instructions and to indicate how a particular instruction is to be written. Table 6–3 explains the notations used in the instruction set descriptions (Section 6.3, page 6-22).

Table 6–3. Instruction Set Descriptions Notations

Notation	Meaning
$\overline{x}$	Logical inversion (1s complement) of x
| x |	Absolute value of x
{ }	Alternative items, one of which must be entered
nnh	Indicates that nn represents a hexadecimal number
(r)	The content of register or location r. *Example:* (dma) means: The value at data memory address dma.
x→y	Value x is assigned to register or location y. *Example:* (dma) → ACC means: The content of the data memory address is put into the accumulator.
x ↔ y	Value x is switched with value y. *Example:* (ACCB) ↔ (ACC) means: The content of the accumulator buffer is switched with the content of the accumulator.
r(n–m)	Bits n through m of register or location r. *Example:* ACC(15–0) means: Bits 15 through 0 of the accumulator.
(r(n–m))	The content of bits n through m of register or location r. *Example:* (ACC(31:16)) means: The content of bits 31 through 16 of the accumulator.

Table 6–3. Instruction Set Descriptions Notations (Continued)

Notation	Meaning
Boldface Characters	Boldface characters in an instruction syntax are to be typed as shown. *Example:* For the syntax: **ADD** *dma*, **16**, you may use a variety of values for *dma*, but the word ADD and the number 16 should be typed as shown. Samples with this syntax follow: **ADD 7h, 16** **ADD X, 16**
italic symbols	Italic symbols in an instruction syntax represent variables. *Example:* For the syntax: **ADD** *dma*, you may use a variety of values for *dma*. Samples with this syntax follow: **ADD DAT** **ADD 15**
#	The # symbol is a prefix for constants used in immediate addressing. For short- or long-immediate operands, it is used in instructions where there is ambiguity with other addressing modes. *Example:* **RPT #15** uses short immediate addressing. It causes the next instruction to be repeated 16 times. **RPT 15** uses direct addressing. The number of times the next instruction repeats is determined by a value stored in memory.
[,x]	Operand x is optional. *Example:* For the syntax: **ADD** *dma*, [,*shift*], you may use a variety of values for *dma*. Samples with this syntax follow: **ADD 7h** You have the option of adding a *shift* value, as in the instruction: **ADD 7h, 5**
[,x1 [,x2]]	Operands x1 and x2 are optional, but you cannot include x2 without also including x1. *Example:* For the syntax: **ADD** *ind*, [,*shift* [,**AR***n*]], you must supply *ind*, as in the instruction: **ADD *+** You have the option of including *shift*, as in the instruction: **ADD *+, 5** If you wish to include **AR***n*, you must also include *shift*, as in: **ADD *+, 0, AR2**

6.2 Instruction Set Summary

This section summarizes the instruction set and instruction set opcodes for the ’C5x. Table 6–4 through Table 6–10 alphabetically list the ’C5x instructions within the following functional groups:

- Accumulator memory reference instructions (Table 6–4)
- Auxiliary registers and data memory page pointer instructions (Table 6–5 on page 6-13)
- Parallel logic unit (PLU) instructions (Table 6–6 on page 6-14)
- TREG0, PREG, and multiply instructions (Table 6–7 on page 6-15)
- Branch and call instructions (Table 6–8 on page 6-17)
- I/O and data memory operation instructions (Table 6–9 on page 6-19)
- Control instructions (Table 6–10 on page 6-20)

The number of words that an instruction occupies in program memory is specified in the Words column of the table. Several instructions specify two values in the Words column because different forms of the instruction occupy a different number of words. For example, the ADD instruction occupies one word when the operand is a short immediate value or two words if the operand is a long immediate value. The number of cycles that an instruction requires to execute is in the Cycles column of the table. The tables assume that all instructions are executed from internal program memory (ROM) and internal data memory (RAM). The cycle timings are for single-instruction execution, not for repeat mode. Additional information is presented in Section 6.3, *Instruction Set Descriptions* on page 6-22. **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

A read or write access to any peripheral memory-mapped register in data memory locations 20h–4Fh will add one cycle to the cycle time shown. This occurs because all peripherals perform these accesses over the TI Bus, which requires an additional cycle.

Note that all writes to external memory require two cycles. Reads require one cycle. Any write access immediately before or after a read cycle will require three cycles (refer to Chapter 8). In addition, if two pipelined instructions try to access the same 2K-word single-access memory block simultaneously, one

extra cycle is required. For example, the DMOV instruction when used with the RPT instruction, requires one cycle in the dual-access RAM but requires two cycles in the single-access RAM. Wait states are added to all external accesses according to the configuration of the software wait-state registers described in Section 9.4, *Software-Programmable Wait-State Generators*, on page 9-13.

Table 6–4. Accumulator Memory Reference Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
ABS	Absolute value of ACC; zero carry bit	1	1	1011	1110	0000	0000	6-28
ADCB	Add ACCB and carry bit to ACC	1	1	1011	1110	0001	0001	6-30
ADD	Add data memory value, with left shift, to ACC	1	1	0010	SHFT	IAAA	AAAA	6-31
	Add data memory value, with left shift of 16, to ACC	1	1	0110	0001	IAAA	AAAA	6-31
	Add short immediate to ACC	1	1	1011	1000	kkkk	kkkk	6-31
	Add long immediate, with left shift, to ACC	2	2	1011	1111	1001	SHFT	6-31
				+ 1 word				
ADDB	Add ACCB to ACC	1	1	1011	1110	0001	0000	6-35
ADDC	Add data memory value and carry bit to ACC with sign extension suppressed	1	1	0110	0000	IAAA	AAAA	6-36
ADDS	Add data memory value to ACC with sign extension suppressed	1	1	0110	0010	IAAA	AAAA	6-38
ADDT	Add data memory value, with left shift specified by TREG1, to ACC	1	1	0110	0011	IAAA	AAAA	6-40
AND	AND data memory value with ACCL; zero ACCH	1	1	0110	1110	IAAA	AAAA	6-43
	AND long immediate, with left shift, with ACC	2	2	1011	1111	1011	SHFT	6-43
				+ 1 word				
	AND long immediate, with left shift of 16, with ACC	2	2	1011	1110	1000	0001	6-43
				+ 1 word				
ANDB	AND ACCB with ACC	1	1	1011	1110	0001	0010	6-46
BSAR	Barrel-shift ACC right	1	1	1011	1111	1110	SHFT	6-82

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
§ Peripheral memory-mapped register access

Table 6–4. Accumulator Memory Reference Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
CMPL	1s complement ACC	1	1	1011	1110	0000	0001	6-94
CRGT	Store ACC in ACCB if ACC > ACCB	1	1	1011	1110	0001	1011	6-100
CRLT	Store ACC in ACCB if ACC < ACCB	1	1	1011	1110	0001	1100	6-102
EXAR	Exchange ACCB with ACC	1	1	1011	1110	0001	1101	6-106
LACB	Load ACC to ACCB	1	1	1011	1110	0001	1111	6-113
LACC	Load data memory value, with left shift, to ACC	1	1	0001	SHFT	IAAA	AAAA	6-114
	Load long immediate, with left shift, to ACC	2	2	1011	1111	1000	SHFT	6-114
				+ 1 word				
	Load data memory value, with left shift of 16, to ACC	1	1	0110	1010	IAAA	AAAA	6-114
LACL	Load data memory value to ACCL; zero ACCH	1	1	0110	1001	IAAA	AAAA	6-117
	Load short immediate to ACCL; zero ACCH	1	1	1011	1001	kkkk	kkkk	6-117
LACT	Load data memory value, with left shift specified by TREG1, to ACC	1	1	0110	1011	IAAA	AAAA	6-120
LAMM	Load contents of memory-mapped register to ACCL; zero ACCH	1	1 or 2§	0000	1000	IAAA	AAAA	6-122
NEG	Negate (2s complement) ACC	1	1	1011	1110	0000	0010	6-177
NORM	Normalize ACC	1	1	1010	0000	IAAA	AAAA	6-181
OR	OR data memory value with ACCL	1	1	0110	1101	IAAA	AAAA	6-187
	OR long immediate, with left shift, with ACC	2	2	1011	1111	1100	SHFT	6-187
				+ 1 word				
	OR long immediate, with left shift of 16, with ACC	2	2	1011	1110	1000	0010	6-187
				+ 1 word				

† **Bold** typeface indicates instructions that are new for the 'C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
§ Peripheral memory-mapped register access

Table 6–4. Accumulator Memory Reference Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
ORB	OR ACCB with ACC	1	1	1011	1110	0001	0011	6-190
ROL	Rotate ACC left 1 bit	1	1	1011	1110	0000	1100	6-210
ROLB	Rotate ACCB and ACC left 1 bit	1	1	1011	1110	0001	0100	6-211
ROR	Rotate ACC right 1 bit	1	1	1011	1110	0000	1101	6-212
RORB	Rotate ACCB and ACC right 1 bit	1	1	1011	1110	0001	0101	6-213
SACB	Store ACC in ACCB	1	1	1011	1110	0001	1110	6-220
SACH	Store ACCH, with left shift, in data memory location	1	1	1001	1SHF	IAAA	AAAA	6-221
SACL	Store ACCL, with left shift, in data memory location	1	1	1001	0SHF	IAAA	AAAA	6-223
SAMM	Store ACCL in memory-mapped register	1	1 or 2§	1000	1000	IAAA	AAAA	6-225
SATH	Barrel-shift ACC right 0 or 16 bits as specified by TREG1	1	1	1011	1110	0101	1010	6-229
SATL	Barrel-shift ACC right as specified by TREG1	1	1	1011	1110	0101	1011	6-231
SBB	Subtract ACCB from ACC	1	1	1011	1110	0001	1000	6-232
SBBB	Subtract ACCB and logical inversion of carry bit from ACC	1	1	1011	1110	0001	1001	6-233
SFL	Shift ACC left 1 bit	1	1	1011	1110	0000	1001	6-237
SFLB	Shift ACCB and ACC left 1 bit	1	1	1011	1110	0001	0110	6-238
SFR	Shift ACC right 1 bit	1	1	1011	1110	0000	1010	6-239
SFRB	Shift ACCB and ACC right 1 bit	1	1	1011	1110	0001	0111	6-241

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
§ Peripheral memory-mapped register access

Table 6–4. Accumulator Memory Reference Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
SUB	Subtract data memory value, with left shift, from ACC	1	1	0011	SHFT	IAAA	AAAA	6-259
	Subtract data memory value, with left shift of 16, from ACC	1	1	0110	0101	IAAA	AAAA	6-259
	Subtract short immediate from ACC	1	1	1011	1010	kkkk	kkkk	6-259
	Subtract long immediate, with left shift, from ACC	2	2	1011 + 1 word	1111	1010	SHFT	6-259
SUBB	Subtract data memory value and logical inversion of carry bit from ACC with sign extension suppressed	1	1	0110	0100	IAAA	AAAA	6-263
SUBC	Conditional subtract	1	1	0000	1010	IAAA	AAAA	6-265
SUBS	Subtract data memory value from ACC with sign extension suppressed	1	1	0110	0110	IAAA	AAAA	6-267
SUBT	Subtract data memory value, with left shift specified by TREG1, from ACC	1	1	0110	0111	IAAA	AAAA	6-269
XOR	Exclusive-OR data memory value with ACCL	1	1	0110	1100	IAAA	AAAA	6-280
	Exclusive-OR long immediate, with left shift of 16, with ACC	2	2	1011 + 1 word	1110	1000	0011	6-280
	Exclusive-OR long immediate, with left shift, with ACC	2	2	1011 + 1 word	1111	1101	SHFT	6-280
XORB	Exclusive-OR ACCB with ACC	1	1	1011	1110	0001	1010	6-283
ZALR	Zero ACCL and load ACCH with rounding	1	1	0110	1000	IAAA	AAAA	6-287
ZAP	Zero ACC and PREG	1	1	1011	1110	0101	1001	6-289

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
§ Peripheral memory-mapped register access

Table 6–5. Auxiliary Registers and Data Memory Page Pointer Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
ADRK	Add short immediate to AR	1	1	0111	1000	kkkk	kkkk	6-42
CMPR	Compare AR with ARCR as specified by CM bits	1	1	1011	1111	0100	01CM	6-95
LAR	Load data memory value to ARx	1	2	0000	0ARX	IAAA	AAAA	6-124
	Load short immediate to ARx	1	2	1011	0ARX	kkkk	kkkk	6-124
	Load long immediate to ARx	2	2	1011	1111	0000	1ARX	6-124
				+ 1 word				
LDP	Load data memory value to DP bits	1	2	0000	1101	IAAA	AAAA	6-127
	Load short immediate to DP bits	1	2	1011	110I	kkkk	kkkk	6-127
MAR	Modify AR	1	1	1000	1011	IAAA	AAAA	6-166
SAR	Store ARx in data memory location	1	1	1000	0ARX	IAAA	AAAA	6-227
SBRK	Subtract short immediate from AR	1	1	0111	1100	kkkk	kkkk	6-234

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

Table 6–6. Parallel Logic Unit (PLU) Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
APL	AND data memory value with DBMR, and store result in data memory location	1	1	0101 1010 IAAA AAAA	6-48
	AND data memory value with long immediate and store result in data memory location	2	2	0101 1110 IAAA AAAA + 1 word	6-48
CPL	Compare data memory value with DBMR	1	1	0101 1011 IAAA AAAA	6-97
	Compare data memory value with long immediate	2	2	0101 1111 IAAA AAAA + 1 word	6-97
OPL	OR data memory value with DBMR and store result in data memory location	1	1	0101 1001 IAAA AAAA	6-184
	OR data memory value with long immediate and store result in data memory location	2	2	0101 1101 IAAA AAAA + 1 word	6-184
SPLK	Store long immediate in data memory location	2	2	1010 1110 IAAA AAAA + 1 word	6-251
XPL	Exclusive-OR data memory value with DBMR and store result in data memory location	1	1	0101 1000 IAAA AAAA	6-284
	Exclusive-OR data memory value with long immediate and store result in data memory location	2	2	0101 1100 IAAA AAAA + 1 word	6-284
LPH	Load data memory value to PREG high byte	1	1	0111 0101 IAAA AAAA	6-133
LT	Load data memory value to TREG0	1	1	0111 0011 IAAA AAAA	6-138

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

Table 6–7. TREG0, PREG, and Multiply Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
LTA	Load data memory value to TREG0; add PREG, with shift specified by PM bits, to ACC	1	1	0111 0000 IAAA AAAA	6-140
LTD	Load data memory value to TREG0; add PREG, with shift specified by PM bits, to ACC; and move data	1	1	0111 0010 IAAA AAAA	6-142
LTP	Load data memory value to TREG0; store PREG, with shift specified by PM bits, in ACC	1	1	0111 0001 IAAA AAAA	6-145
LTS	Load data memory value to TREG0; subtract PREG, with shift specified by PM bits, from ACC	1	1	0111 0100 IAAA AAAA	6-147
MAC	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by program memory value and store result in PREG	2	3	1010 0010 IAAA AAAA + 1 word	6-149
MACD	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by program memory value and store result in PREG; and move data	2	3	1010 0011 IAAA AAAA + 1 word	6-153
MADD	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG; and move data	1	3	1010 1011 IAAA AAAA	6-158
MADS	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG	1	3	1010 1010 IAAA AAAA	6-162

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

Table 6–7. TREG0, PREG, and Multiply Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
MPY	Multiply data memory value by TREG0 and store result in PREG	1	1	0101	0100	IAAA	AAAA	6-168
	Multiply short immediate by TREG0 and store result in PREG	1	1	110k	kkkk	kkkk	kkk	6-168
	Multiply long immediate by TREG0 and store result in PREG	2	2	1011 + 1 word	1110	1000	0000	6-168
MPYA	Add PREG, with shift specified by PM bits, to ACC; multiply data memory value by TREG0 and store result in PREG	1	1	0101	0000	IAAA	AAAA	6-171
MPYS	Subtract PREG, with shift specified by PM bits, from ACC; multiply data memory value by TREG0 and store result in PREG	1	1	0101	0001	IAAA	AAAA	6-173
MPYU	Multiply unsigned data memory value by TREG0 and store result in PREG	1	1	0101	0101	IAAA	AAAA	6-175
PAC	Load PREG, with shift specified by PM bits, to ACC	1	1	1011	1110	0000	0011	6-193
SPAC	Subtract PREG, with shift specified by PM bits, from ACC	1	1	1011	1110	0000	0101	6-246
SPAC	Subtract PREG, with shift specified by PM bits, from ACC	1	1	1011	1110	0000	0101	6-246
SPH	Store PREG high byte, with shift specified by PM bits, in data memory location	1	1	1000	1101	IAAA	AAAA	6-247
SPL	Store PREG low byte, with shift specified by PM bits, in data memory location	1	1	1000	1100	IAAA	AAAA	6-249
SPM	Set product shift mode (PM) bits	1	1	1011	1111	0000	00PM	6-252
SQRA	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; square value and store result in PREG	1	1	0101	0010	IAAA	AAAA	6-253

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

Table 6–7. TREG0, PREG, and Multiply Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
SQRS	Subtract PREG, with shift specified by PM bits, from ACC; load data memory value to TREG0; square value and store result in PREG	1	1	0101 0011 IAAA AAAA	6-255
ZPR	Zero PREG	1	1	1011 1110 0101 1000	6-290

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

Table 6–8. Branch and Call Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
B	Branch unconditionally to program memory location	2	4	0111 1001 1AAA AAAA + 1 word	6-51
BACC	Branch to program memory location specified by ACCL	1	4	1011 1110 0010 0000	6-52
BACCD	Delayed branch to program memory location specified by ACCL	1	2	1011 1110 0010 0001	6-53
BANZ	Branch to program memory location if AR not zero	2	4¶ or 2#	0111 1011 1AAA AAAA + 1 word	6-54
BANZD	Delayed branch to program memory location if AR not zero	2	2	0111 1111 1AAA AAAA + 1 word	6-56
BCND	Branch conditionally to program memory location	2	4¶ or 2#	1110 00TP ZLVC ZLVC + 1 word	6-58
BCNDD	Delayed branch conditionally to program memory location	2	2	1111 00TP ZLVC ZLVC + 1 word	6-60
BD	Delayed branch unconditionally to program memory location	2	2	0111 1101 1AAA AAAA + 1 word	6-62
CALA	Call to subroutine addressed by ACCL	1	4	1011 1110 0011 0000	6-83

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
¶ Conditions true
# Condition false

Table 6–8. Branch and Call Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
CALAD	Delayed call to subroutine addressed by ACCL	1	2	1011 1110 0011 1101	6-84
CALL	Call to subroutine unconditionally	2	4	0111 1010 1AAA AAAA + 1 word	6-85
CALLD	Delayed call to subroutine unconditionally	2	2	0111 1110 1AAA AAAA + 1 word	6-86
CC	Call to subroutine conditionally	2	4¶ or 2#	1110 10TP ZLVC ZLVC + 1 word	6-88
CCD	Delayed call to subroutine conditionally	2	2	1111 10TP ZLVC ZLVC + 1 word	6-90
INTR	Software interrupt that branches program control to program memory location	1	4	1011 1110 011I NTR#	6-111
NMI	Nonmaskable interrupt and globally disable interrupts (INTM = 1)	1	4	1011 1110 0101 0010	6-179
RET	Return from subroutine	1	4	1110 1111 0000 0000	6-202
RETC	Return from subroutine conditionally	1	2	1110 11TP ZLVC ZLVC	6-203
RETCD	Delayed return from subroutine conditionally	1	4¶ or 2#	1111 11TP ZLVC ZLVC	6-205
RETD	Delayed return from subroutine	1	2	1111 1111 0000 0000	6-207
RETE	Return from interrupt with context switch and globally enable interrupts (INTM = 0)	1	4	1011 1110 0011 1010	6-208
RETI	Return from interrupt with context switch	1	4	1011 1110 0011 1000	6-209
TRAP	Software interrupt that branches program control to program memory location 22h	1	4	1011 1110 0101 0001	6-277
XC	Execute next instruction(s) conditionally	1	1	111N 01TP ZLVC ZLVC	6-278

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
¶ Conditions true
# Condition false

Table 6–9. I/O and Data Memory Operation Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
BLDD	Block move from data to data memory	2	3	1010 1000 IAAA AAAA + 1 word	6-67
	Block move from data to data memory with destination address long immediate	2	3	1010 1001 IAAA AAAA + 1 word	6-67
	Block move from data to data memory with source address in BMAR	1	2	1010 1100 IAAA AAAA	6-67
	Block move from data to data memory with destination address in BMAR	1	2	1010 1101 IAAA AAAA	6-67
BLDP	Block move from data to program memory with destination address in BMAR	1	2	0101 0111 IAAA AAAA	6-73
BLPD	Block move from program to data memory with source address in BMAR	1	2	1010 0100 IAAA AAAA	6-76
	Block move from program to data memory with source address long immediate	2	3	1010 0101 IAAA AAAA + 1 word	6-76
DMOV	Move data in data memory	1	1	0111 0111 IAAA AAAA	6-104
IN	Input data from I/O port to data memory location	2	2	1010 1111 IAAA AAAA + 1 word	6-109
LMMR	Load data memory value to memory-mapped register	2	2 or 3§	1000 1001 IAAA AAAA + 1 word	6-130
OUT	Output data from data memory location to I/O port	2	3	0000 1100 IAAA AAAA + 1 word	6-191
SMMR	Store memory-mapped register in data memory location	2	2 or 3§	0000 1001 IAAA AAAA + 1 word	6-243
TBLR	Transfer data from program to data memory with source address in ACCL	1	3	1010 0110 IAAA AAAA	6-271
TBLW	Transfer data from data to program memory with destination address in ACCL	1	3	1010 0111 IAAA AAAA	6-274

† **Bold** typeface indicates instructions that are new for the 'C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.
§ Peripheral memory-mapped register access

Table 6–10. Control Instructions

Mnemonic†	Description	Words	Cycles‡	Opcode				Page
BIT	Test bit	1	1	0100	BITX	IAAA	AAAA	6-63
BITT	Test bit specified by TREG2	1	1	0110	1111	IAAA	AAAA	6-65
CLRC	Clear overflow mode (OVM) bit	1	1	1011	1110	0100	0010	6-92
	Clear sign extension mode (SXM) bit	1	1	1011	1110	0100	0110	6-92
	Clear hold mode (HM) bit	1	1	1011	1110	0100	1000	6-92
	Clear test/control (TC) bit	1	1	1011	1110	0100	1010	6-92
	Clear carry (C) bit	1	1	1011	1110	0100	1110	6-92
	Clear configuration control (CNF) bit	1	1	1011	1110	0100	0100	6-92
	Clear interrupt mode (INTM) bit	1	1	1011	1110	0100	0000	6-92
	Clear external flag (XF) pin	1	1	1011	1110	0100	1100	6-92
IDLE	Idle until nonmaskable interrupt or reset	1	1	1011	1110	0010	0010	6-107
IDLE2	Idle until nonmaskable interrupt or reset — low-power mode	1	1	1011	1110	0010	0011	6-108
LST	Load data memory value to ST0	1	2	0000	1110	IAAA	AAAA	6-135
	Load data memory value to ST1	1	2	0000	1111	IAAA	AAAA	6-135
NOP	No operation	1	1	1000	1011	0000	0000	6-180
POP	Pop top of stack to ACCL; zero ACCH	1	1	1011	1110	0011	0010	6-194
POPD	Pop top of stack to data memory location	1	1	1000	1010	IAAA	AAAA	6-196
PSHD	Push data memory value to top of stack	1	1	0111	0110	IAAA	AAAA	6-198
PUSH	Push ACCL to top of stack	1	1	1011	1110	0011	1100	6-200

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

Table 6–10. Control Instructions (Continued)

Mnemonic†	Description	Words	Cycles‡	Opcode	Page
RPT	Repeat next instruction specified by data memory value	1	1	0000 1011 IAAA AAAA	6-214
	Repeat next instruction specified by short immediate	1	2	1011 1011 kkkk kkkk	6-214
	Repeat next instruction specified by long immediate	2	2	1011 1110 1100 0100 + 1 word	6-214
RPTB	Repeat block of instructions specified by BRCR	2	2	1011 1110 1100 0110 + 1 word	6-217
RPTZ	Clear ACC and PREG; repeat next instruction specified by long immediate	2	2	1011 1110 1100 0101 + 1 word	6-219
SETC	Set overflow mode (OVM) bit	1	1	1011 1110 0100 0011	6-235
	Set sign extension mode (SXM) bit	1	1	1011 1110 0100 0111	6-235
	Set hold mode (HM) bit	1	1	1011 1110 0100 1001	6-235
	Set test/control (TC) bit	1	1	1011 1110 0100 1011	6-235
	Set carry (C) bit	1	1	1011 1110 0100 1111	6-235
	Set external flag (XF) pin high	1	1	1011 1110 0100 1101	6-235
	Set configuration control (CNF) bit	1	1	1011 1110 0100 0101	6-235
	Set interrupt mode (INTM) bit	1	1	1011 1110 0100 0001	6-235
SST	Store ST0 in data memory location	1	1	1000 1110 IAAA AAAA	6-257
	Store ST1 in data memory location	1	1	1000 1111 IAAA AAAA	6-257

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ The cycle timings are for single-instruction execution, not for repeat mode.

6.3 Instruction Set Descriptions

This section provides detailed information on the instruction set for the ’C5x family; see Table 6–4 through Table 6–10 for a complete list of available instructions. Each instruction description presents the following information:

- ❏ Assembler syntax
- ❏ Operands
- ❏ Opcodes
- ❏ Execution
- ❏ Status Bits
- ❏ Description
- ❏ Words
- ❏ Cycles
- ❏ Examples

The **EXAMPLE** instruction is provided to familiarize you with the format of the instruction descriptions and to explain what is described under each heading.

Syntax

Direct: **EXAMPLE** *dma* [,*shift*]
Indirect: **EXAMPLE** {*ind*} [,*shift*] [,**AR***n*]
Short immediate: **EXAMPLE** #*k*
Long immediate: **EXAMPLE** #*lk*

Each instruction description begins with an assembly language syntax expression. A source statement can contain four ordered fields. The general syntax for source statements is as follows:

[*label*] [:] *mnemonic* [*operand list*] [***;****comment*]

Follow these guidelines:

- ❏ All statements must begin with a label, a blank, an asterisk, or a semicolon.
- ❏ Labels are optional; if used, they must begin in column 1. Labels may be placed either before the instruction mnemonic on the same line or on the preceding line in the first column.
- ❏ One or more blanks must separate each field. Tab characters are equivalent to blanks.
- ❏ Comments are optional. Comments that begin in column 1 can begin with an asterisk or a semicolon (* or ;), but comments that begin in any other column **mus**t begin with a semicolon.

See Table 6–2 on page 6-4 for definitions of symbols and abbreviations used in the syntax expression.

Operands

$0 \le dma \le 127$
$0 \le pma \le 65535$
$0 \le shift \le 15$
$0 \le shift2 \le 7$
$0 \le n \le 7$
$0 \le k \le 255$
$0 \le lk \le 65535$
$0 \le x \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Operands can be constants or assembly-time expressions that refer to memory, I/O ports, register addresses, pointers, shift counts, and a variety of other constants. This section also gives the range of acceptable values for the operand types.

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

The opcode graphic shows bit values or field names that make up each instruction. See Table 6–1 on page 6-2 for definitions of symbols and abbreviations used in the instruction opcodes.

Execution

(PC) + 1 → PC
(ACC) + (dma) → ACC
0 → C

The execution section symbolically represents the process that takes place when the instruction is executed. See Table 6–2 on page 6-4 for definitions of symbols and abbreviations used in the execution section.

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM	SXM	C and OV

An instruction's execution may be affected by the state of the fields in the status registers; also it may affect the state of the status register fields. Both the effects *on* and the effects *of* the status register fields are listed in this section.

Description

This section describes the instruction execution and its effect on the rest of the processor or memory contents. Any constraints on the operands imposed by the processor or the assembler are discussed. The description parallels and supplements the information given symbolically in the execution section.

Words

This section specifies the number of memory words required to store the instruction and its extension words.

Cycles

This section provides tables showing the number of cycles required for a given instruction to execute in a given memory configuration — both as a single instruction and in the repeat (RPT) mode. The following are examples of the cycle timing tables.

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

The column headings in the tables indicate the program source location. The program source locations are defined as follows:

ROM	The instruction executes from on-chip program ROM.
DARAM	The instruction executes from on-chip dual-access program RAM.
SARAM	The instruction executes from on-chip single-access program RAM.
External Memory	The instruction executes from external program memory.

If an instruction requires memory operand(s), the rows in the tables indicate the location(s) of the operand(s). The operands are defined as follows:

DARAM	The operand is in internal dual-access RAM.
SARAM	The operand is in internal single-access RAM.
External	The operand is in external memory.
ROM	The operand is in internal program ROM.
MMR	The operand is a memory-mapped register.
MMPORT	The operand is a memory-mapped I/O port.

The number of cycles required for each instruction is given in terms of the processor machine cycles (CLKOUT1 period). The additional wait states for program/data memory and I/O accesses are defined below. Note that these additional cycles can be generated by the on-chip software wait-state generator or by the external READY signal. These variables can also use the subscripts *src, dst,* and *code* to indicate source, destination, and code, respectively.

d Data memory wait states. Represents the number of additional clock cycles the device waits for external data memory to respond to an access.

io I/O wait states. Represents the number of additional clock cycles the device waits for an external I/O to respond to an access.

n Repetitions (where $n > 2$ to fill the pipeline). Represents the number of times a repeated instruction is executed.

p Program memory wait states. Represents the number of additional clock cycles the device waits for external program memory to respond to an access.

Table 6–11 lists the on-chip single-access RAM available on each ’C5x processor. The on-chip single-access RAM is divided into 1K- and/or 2K-word blocks contiguous in address memory space. All ’C5x processors support parallel accesses to these on-chip SARAM blocks. However, one SARAM block allows only one access per cycle. In other words, the processor can read/write on one SARAM block while accessing another SARAM block.

All external reads require at least one machine cycle while all external writes require at least two machine cycles. However, if an external write is immediately followed or preceded by an external read cycle, then the external write requires three cycles. See Section 8.9, *External Memory Interface Timings*, on page 8-39 for details. If you use an on-chip wait-state generator to add *m* (*m*>0) wait states to an external access, then both the external reads and the external writes require *m*+1 cycles, assuming that the external READY line is driven high. If you use the READY input line to add *m* additional cycles to an external access, then external reads require *m*+1 cycles and external write accesses require *m*+2 cycles. See Section 9.4, *Software-Programmable Wait-State Generators*, on page 9-13 and the data sheet for READY electrical specifications.

Table 6–11. Address Blocks for On-Chip Single-Access RAM

Device	SARAM	Block size	Hex Address Range
’C50	9K-word	2K-word block	0800–0FFF
		2K-word block	1000–17FF
		2K-word block	1800–1FFF
		2K-word block	2000–27FF
		1K-word block	2800–2BFF
’C51	1K-word	1K-word block	0800–0BFF
’C53/’C53S	3K-word	2K-word block	0800–0FFF
		1K-word block	1000–13FF
’LC56	6K-word	2K-word block	0800–0FFF
		2K-word block	1000–17FF
		2K-word block	1800–1FFF
’C57S/’LC57	6K-word	2K-word block	0800–0FFF
		2K-word block	1000–17FF
		2K-word block	1800–1FFF

The instruction cycles are based on the following assumptions:

- ❑ At least four instructions following the current instruction are fetched from the same memory section (on-chip or external) as the current instruction, except in instructions that cause a program counter discontinuity, such as B, CALL, etc.
- ❑ When executing a single instruction, there is no pipeline conflict between the current instruction and the instructions immediately preceding or following that instruction. The only exception is the conflict between the fetch phase of the pipeline and the memory read/write (if any) access of the instruction under consideration. See Chapter 7 for pipeline operation.
- ❑ In the repeat execution mode, all conflicts caused by the pipelined execution of that instruction are considered.

Refer to Appendix B for a summary of instruction cycle classifications.

Example

Example code is shown for each instruction. The effect of the code on memory and/or registers is summarized.

Syntax **ABS**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0

Execution

(PC) + 1 → PC
|(ACC)| → ACC
0 → C

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM	SXM	C and OV

Description If the contents of the accumulator (ACC) are greater than or equal to 0, the contents of the ACC is unchanged. If the contents of the ACC are less than 0, the contents of the ACC is replaced by its 2s-complement value. The ABS instruction clears the C bit.

Note that 8000 0000h is a special case. When the OVM bit is cleared, the ABS of 8000 0000h is 8000 0000h. When the OVM bit is set, the ABS of 8000 0000h is 7FFF FFFFh. In either case, the OV bit is set.

ABS is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

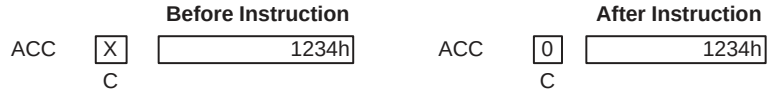
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

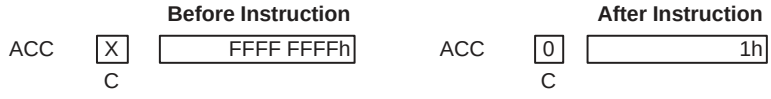
Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1 ABS

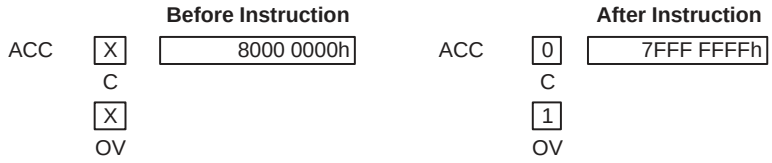


Example 2 ABS



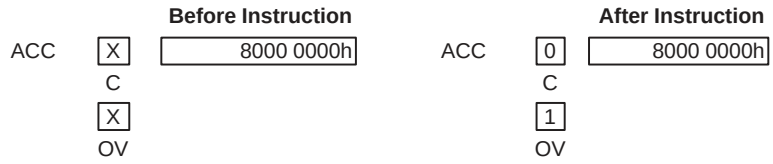
Example 3

```
ABS ;(OVM = 1)
```



Example 4

```
ABS ;(OVM = 0)
```



Syntax **ADCB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	0	1

Execution

(PC) + 1 → PC
(ACC) + (ACCB) + (C) → ACC

Status Bits

Affected by:	*Affects*:
OVM	C and OV

Description The contents of the accumulator buffer (ACCB) and the value of the C bit are added to the contents of the accumulator (ACC). The result is stored in the ACC and the contents of the ACCB are unaffected. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

ADCB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

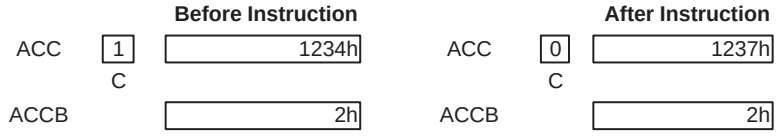
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
ADCB
```



Syntax

Direct: **ADD** *dma* [,*shift*]
Indirect: **ADD** {i*nd*} [,*shift*] [,**AR***n*]
Short immediate: **ADD** #*k*
Long immediate: **ADD** #*lk* [,*shift*]

Operands

0 ≤ dma ≤ 127
0 ≤ shift ≤16 (defaults to 0)
0 ≤ n ≤ 7
0 ≤ k ≤ 255
–32768 ≤ lk ≤ 32767
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	SHFT †				0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	SHFT †				1	See Section 5.2						

† See Table 6–1 on page 6-2.

Direct addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	0	dma						

Indirect addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0	8-Bit Constant							

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	0	1	SHFT †			
16-Bit Constant															

† See Table 6–1 on page 6-2.

Execution

Direct or indirect addressing:
(PC) + 1 → PC
(ACC) + ((dma) × 2^{shift}) → ACC

Short immediate addressing:
(PC) + 1 $\rightarrow$ PC
(ACC) + k $\rightarrow$ ACC

Long immediate addressing:
(PC) + 2 $\rightarrow$ PC
(ACC) + (lk $\times$ 2^{shift}) $\rightarrow$ ACC

Status Bits

Affected by:	*Affects*:	
OVM and SXM	C and OV	Direct or indirect addressing
OVM	C and OV	Short immediate addressing
OVM and SXM	C and OV	Long immediate addressing

Description

If direct, indirect, or long immediate addressing is used, the contents of the data memory address (dma) or a 16-bit constant are shifted left, as defined by the shift code, and added to the contents of the accumulator (ACC). The result is stored in the ACC. During shifting, the accumulator low byte (ACCL) is zero-filled. If the SXM bit is cleared, the high-order bits of the ACC are zero-filled; if the SXM bit is set, the high-order bits of the ACC are sign-extended.

Note that when the auxiliary register pointer (ARP) is updated during indirect addressing, you must specify a shift operand. If you don't want a shift, you must enter a 0 for this operand. For example:

```
ADD*+,0,AR0
```

If short immediate addressing is used, an 8-bit positive constant is added to the contents of the ACC. The result is stored in the ACC. In this mode, no shift value may be specified and the addition is unaffected by the SXM bit.

The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared. If a 16-bit shift is specified with the ADD instruction, the C bit is set only if the result of the addition generates a carry; otherwise, the C bit is unaffected. This allows the accumulation to generate the proper single carry when a 32-bit number is added to the ACC.

ADD is an accumulator memory reference instruction (see Table 6–4).

Words

1 (Direct, indirect, or short immediate addressing)

2 (Long immediate addressing)

Cycles

For the short and long immediate addressing modes, the ADD instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

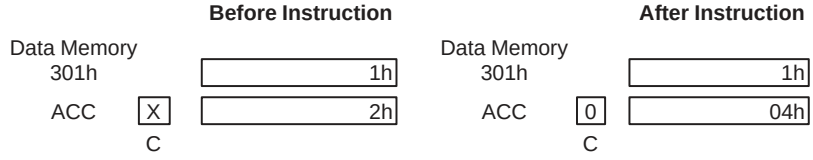
	ROM	DARAM	SARAM	External Memory
	1	1	1	1+p

Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

Example 1

```
ADD  DAT1,1  ;(DP = 6)
```



Example 2

```
ADD  *+,0,AR0
```

	Before Instruction	After Instruction
ARP	4	0
AR4	0302h	0303h
Data Memory 302h	2h	2h
ACC	2h	04h
C	X	0

Example 3

```
ADD  #1h ;Add short immediate
```

	Before Instruction	After Instruction
ACC	2h	03h
C	X	0

Example 4

```
ADD  #1111h,1 ;Add long immediate with shift of 1
```

	Before Instruction	After Instruction
ACC	2h	2224h
C	X	0

Syntax **ADDB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	0	0

Execution

(PC) + 1 → PC
(ACC) + (ACCB) → ACC

Status Bits

Affected by:	*Affects*:
OVM	C and OV

Description

The contents of the accumulator buffer (ACCB) are added to the contents of the accumulator (ACC). The result is stored in the ACC and the contents of the ACCB are unaffected. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

ADDB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

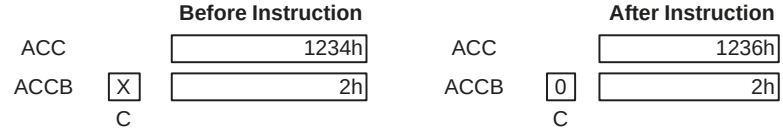
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
ADDB
```



Syntax

Direct: **ADDC** *dma*
Indirect: **ADDC** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(ACC) + (dma) + (C) → ACC

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM	SXM	C and OV

Description

The contents of the data memory address (dma) and the value of the C bit are added to the contents of the accumulator (ACC) with sign extension suppressed. The result is stored in the ACC. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

The ADDC instruction can be used in performing multiple-precision arithmetic. ADDC is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

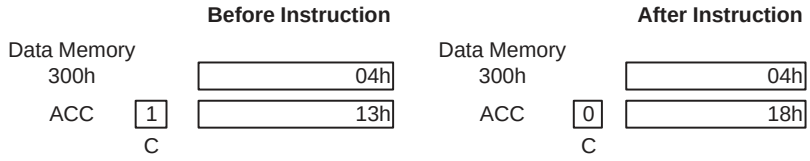
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

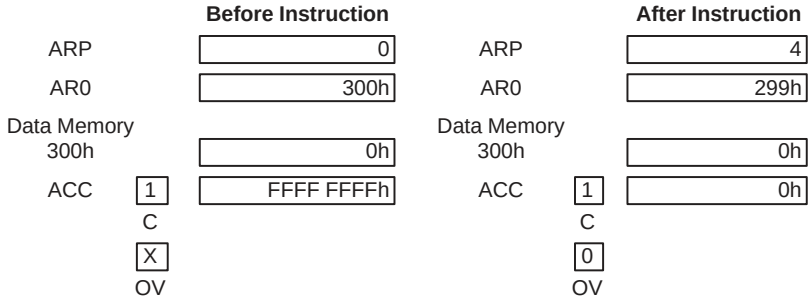
Example 1

```
ADDC DAT0 ;(DP = 6)
```



Example 2

```
ADDC *–,AR4 ;(OVM = 0)
```



Syntax

Direct: **ADDS** *dma*
Indirect: **ADDS** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(ACC) + (dma) → ACC
(dma) is an unsigned16-bit number

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM	SXM	C and OV

Description

The contents of the data memory address (dma) are added to the contents of the accumulator (ACC) with sign extension suppressed. The data is treated as an unsigned 16-bit number, regardless of the SXM bit. The contents of the ACC are treated as a signed number. The result is stored in the ACC. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

The ADDS instruction produces the same results as an ADD instruction with the SXM bit cleared and a shift count of 0.

ADDS is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

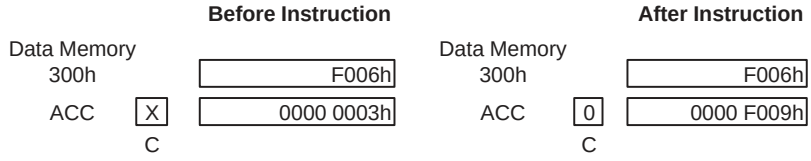
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

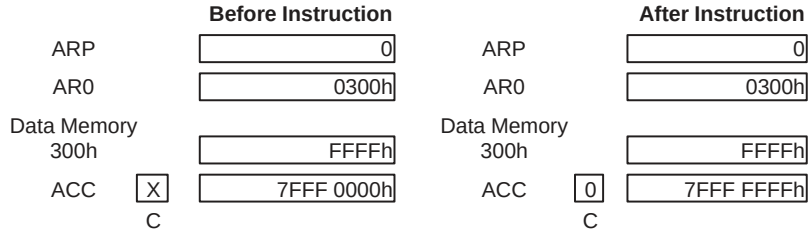
Example 1

```
ADDS DAT0 ;(DP = 6)
```



Example 2

```
ADDS *
```



Syntax

Direct: **ADDT** *dma*
Indirect: **ADDT** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	1	1	See Section 5.2						

Execution

$(PC) + 1 \rightarrow PC$
$(ACC) + ((dma) \times 2^{TREG1(3–0)}) \rightarrow ACC$

If SXM = 0:
(dma) is not sign-extended
If SXM = 1:
(dma) is sign-extended

Status Bits

Affected by:
OVM, SXM, and TRM

Affects:
C and OV

Description

The contents of the data memory address (dma) are shifted left from 0 to 15 bits, as defined by the 4 LSBs of TREG1, and added to the contents of the accumulator (ACC). The result is stored in the ACC. Sign extension on the dma value is controlled by the SXM bit. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs. Subsequent calls to the ADDT instruction will shift the value by the TREG1 value (which is the same as TREG0), maintaining ’C5x object-code compatibility with the ’C2x.

ADDT is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

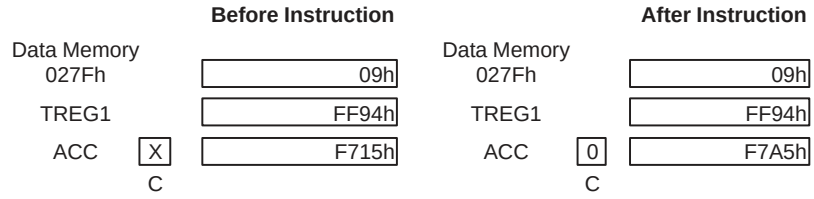
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

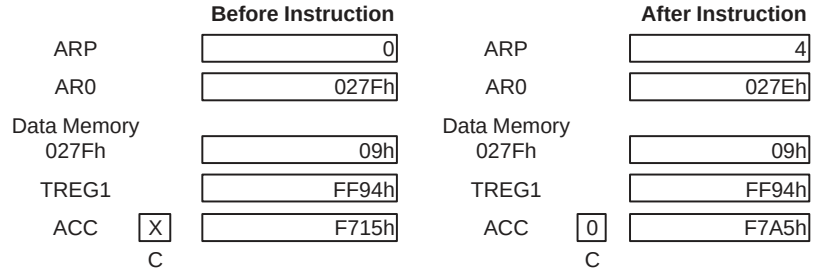
Example 1

```
ADDT DAT127 ;(DP = 4, SXM = 0)
```



Example 2

```
ADDT *–,AR4 ;(SXM = 0)
```



Syntax **ADRK** #*k*

Operands $0 \leq k \leq 255$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	0	8-Bit Constant							

Execution

(PC) + 1 → PC
(current AR) + 8-bit positive constant → current AR

Status Bits None affected.

Description The 8-bit immediate value, right-justified, is added to the current auxiliary register (AR). The result is stored in the AR. The addition takes place in the auxiliary register arithmetic unit (ARAU), with the immediate value treated as an 8-bit positive integer. All arithmetic operations on the AR are unsigned.

ADRK is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words 1

Cycles The ADRK instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Example

```
ADRK #80h
```

	Before Instruction		After Instruction
ARP	5	ARP	5
AR5	4321h	AR5	43A1h

Syntax

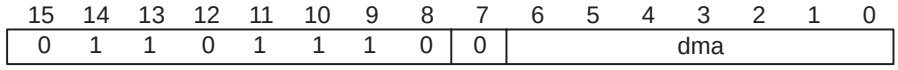
Direct: **AND** *dma*
Indirect: **AND** {*ind*} [,**AR***n*]
Long immediate: **AND** #*lk* [,*shift*]

Operands

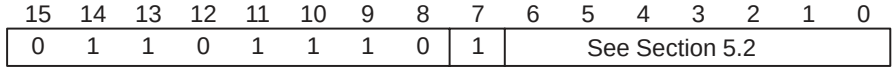
$0 \le dma \le 127$
$0 \le n \le 7$
lk: 16-bit constant
$0 \le shift \le 16$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

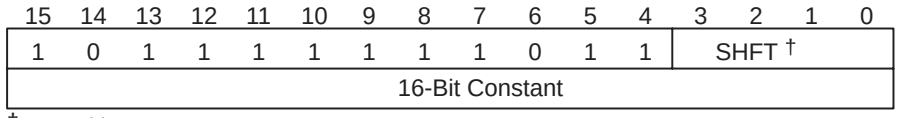
Direct addressing



Indirect addressing

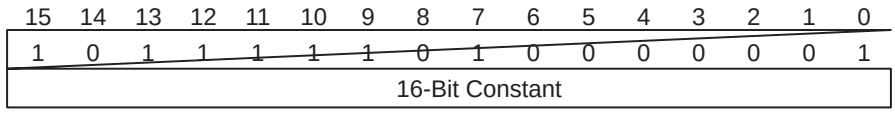


Long immediate addressing with shift



† See Table 6–1 on page 6-2.

Long immediate addressing with shift of 16



Execution

Direct or indirect addressing:
(PC) + 1 → PC
(ACC(15–0)) **AND** (dma) → ACC(15–0)
0 → ACC(31–16)

Long immediate addressing:
(PC) + 2 → PC
(ACC(30–0)) **AND** (lk × 2^{shift}) → ACC

Status Bits

Not affected by:

SXM	Long immediate addressing

Description

If a long immediate constant is specified, the constant is shifted left and zero-extended on both ends and is ANDed with the contents of the accumulator (ACC). The result is stored in the ACC. If a constant is not specified, the contents of the data memory address (dma) are ANDed with the contents of the accumulator low byte (ACCL). The result is stored in the ACCL and the accumulator high byte (ACCH) is zero-filled.

AND is an accumulator memory reference instruction (see Table 6–4).

Words 1 (Direct or indirect addressing)

2 (Long immediate addressing)

Cycles For the long immediate addressing modes, the AND instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block.

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block.

Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

Example 1

```
AND DAT16 ;(DP = 4)
```

	Before Instruction		After Instruction
Data Memory 0210h	00FFh	Data Memory 0210h	00FFh
ACC	1234 5678h	ACC	0000 0078h

Example 2

```
AND *
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	0301h	AR0	0301h
Data Memory 0301h	FF00h	Data Memory 0301h	FF00h
ACC	1234 5678h	ACC	0000 5600h

Example 3 `AND #00FFh,4`

	Before Instruction		After Instruction
ACC	1234 5678h	ACC	0000 0670h

Syntax **ANDB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	0

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) **AND** (ACCB) $\rightarrow$ ACC

Status Bits None affected.

Description The contents of the accumulator (ACC) are ANDed with the contents of the accumulator buffer (ACCB). The result is stored in the ACC and the contents of the ACCB are unaffected.

ANDB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example ANDB

	Before Instruction		After Instruction
ACC	0F0F FFFFh	ACC	0505 5555h
ACCB	5555 5555h	ACCB	5555 5555h

Syntax **APAC**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0

Execution

(PC) + 1 → PC
(ACC) + (shifted PREG) → ACC

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM and PM	SXM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the contents of the accumulator (ACC). The result is stored in the ACC. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared. The contents of the PREG are always sign extended.

The APAC instruction is a subset of the LTA, LTD, MAC, MACD, MADS, MADD, MPYA, and SQRA instructions.

APAC is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

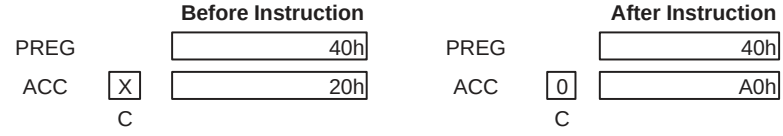
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
APAC ;(PM = 01)
```



Syntax

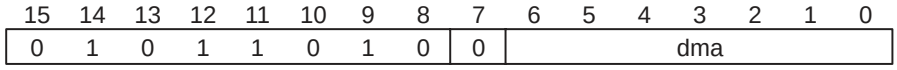
Direct: **APL** [#*lk*,] *dma*
Indirect: **APL** [#*lk*,] {*ind*} [,**AR***n*]

Operands

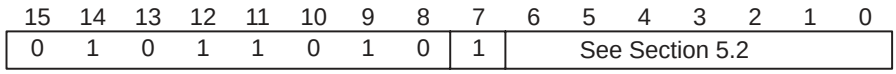
0 ≤ dma ≤ 127
lk: 16-bit constant
0 ≤ n ≤ 7
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

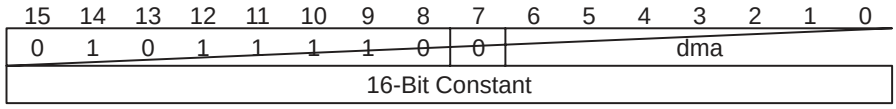
Direct addressing with long immediate not specified



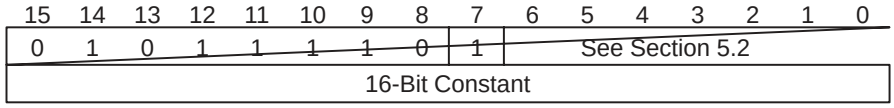
Indirect addressing with long immediate not specified



Direct addressing with long immediate specified



Indirect addressing with long immediate specified



Execution

Long immediate not specified:
(PC) + 1 → PC
(dma) **AND** (DBMR) → dma

Long immediate specified:
(PC) + 2 → PC
(dma) **AND** lk → dma

Status Bits

Affects: TC

Description

If a long immediate constant is specified, the constant is ANDed with the contents of the data memory address (dma). If a constant is not specified, the contents of the dma are ANDed with the contents of the dynamic bit manipulation register (DBMR). In both cases, the result is written directly back to the dma and the contents of the accumulator (ACC) are unaffected. The TC bit is set, if the result of the AND operation is 0; otherwise, the TC bit is cleared.

APL is a parallel logic unit (PLU) instruction (see Table 6–6).

Words

1 (Long immediate not specified)

2 (Long immediate specified)

Cycles

Cycles for a Single Instruction (second operand DBMR)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (second operand DBMR)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n–2	2n–2	2n–2, 2n+1†	2n–2+p
External	4n–2+2nd	4n–2+2nd	4n–2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2	2+2p
External	3+2d	3+2d	3+2d	6+2d+2p

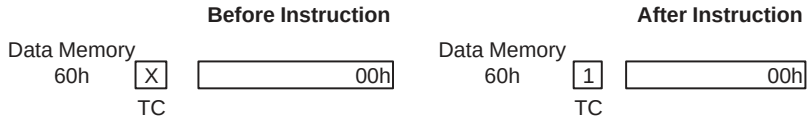
Cycles for a Repeat (RPT) Execution (long immediate specified)

	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	2n–1	2n–1	2n–1, 2n+2†	2n–1+2p
External	4n–1+2nd	4n–1+2nd	4n–1+2nd	4n+2+2nd+2p

† If the operand and the code reside in same SARAM block

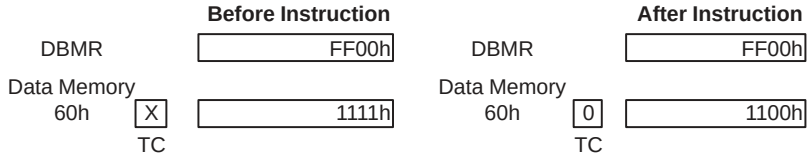
Example 1

```
APL #0023h,DAT96 ;(DP = 0)
```



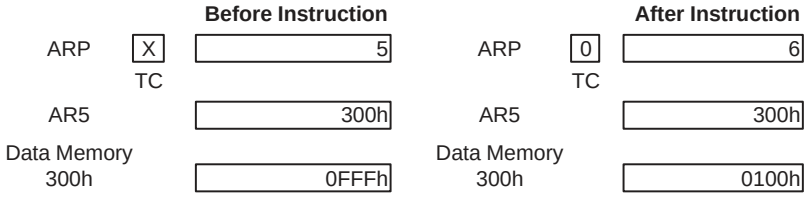
Example 2

```
APL DAT96 ;(DP = 0)
```



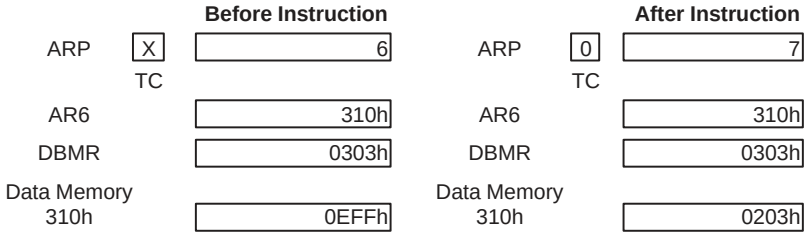
Example 3

```
APL #0100h,*,AR6
```



Example 4

```
APL *,AR7
```



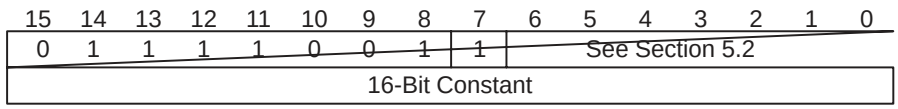
Syntax

B *pma* [, {*ind*} [,**AR***n*]]

Operands

$0 \le pma \le 65535$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode



Execution

pma $\rightarrow$ PC
Modify current AR and ARP as specified

Status Bits

None affected.

Description

Control is passed to the program memory address (pma). The current auxiliary register (AR) and auxiliary register pointer (ARP) are modified as specified. The pma can be either a symbolic or numeric address.

B is a branch and call instruction (see Table 6–8).

Words

2

Cycles

The B instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+4p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
B 191,*+,AR1
```

The value 191 is loaded into the program counter (PC), and the program continues executing from that location. The current AR is incremented by 1, and ARP is set to 1.

Syntax **BACC**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0

Execution ACC(15–0) $\rightarrow$ PC

Status Bits None affected.

Description Control is passed to the 16-bit address residing in the accumulator low byte (ACCL).

BACC is a branch and call instruction (see Table 6–8).

Words 1

Cycles The BACC instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The ’C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
BACC ;(ACC contains the value 191)
```

The value 191 is loaded into the program counter (PC), and the program continues executing from that location.

Syntax **BACCD**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	1

Execution ACC(15–0) $\rightarrow$ PC

Status Bits None affected.

Description The one 2-word instruction or two 1-word instructions following the BACCD instruction are fetched from program memory and executed before the branch is taken. After the instructions are executed, control is passed to the 16-bit address residing in the accumulator low byte (ACCL).

BACCD is a branch and call instruction (see Table 6–8).

Words 1

Cycles The BACCD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

Example

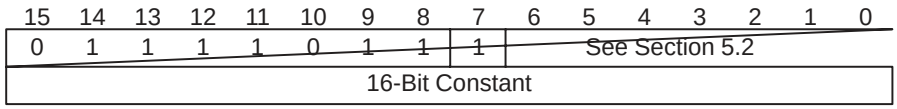
```
BACCD          ;(ACC contains the value 191)
MAR *+,AR1
LDP #5
```

After the current AR, ARP, and DP are modified as specified, program execution continues from location 191.

Syntax **BANZ** *pma* [, {*ind*} [,**AR***n*]]

Operands $0 \le pma \le 65535$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode



Execution
If (current AR) ≠ 0:
    pma → PC
Else:
    (PC) + 2 → PC
Modify current AR as specified

Status Bits None affected.

Description If the contents of the current auxiliary register (AR) are not 0, control is passed to the program memory address (pma); otherwise, control is passed to the next instruction. The default modification to current AR is a decrement by 1. You can cause N loop iterations to be executed by initializing the auxiliary register loop counter to N–1 before loop entry. The pma can be either a symbolic or numeric address.

BANZ is a branch and call instruction (see Table 6–8).

Words 2

Cycles The BANZ instruction is not repeatable.

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+4p†
False	2	2	2	2+2p

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example 1

```
BANZ PGM0
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	5h	AR0	4h

0 is loaded into the program counter (PC), and the program continues executing from that location.

or

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	0h	AR0	FFFFh

The PC is incremented by 2, and execution continues from that location.

Example 2

```
        MAR *,AR0
        LAR AR1,#3
        LAR AR0,#60h
PGM191  ADD *+,AR1
        BANZ PGM191,AR0
```

The contents of data memory locations 60h–63h are added to the accumulator (ACC).

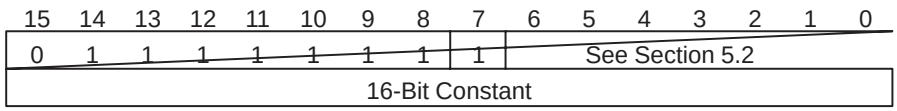
Syntax

BANZD *pma* [, {*ind*} [,**AR***n*]]

Operands

$0 \le pma \le 65535$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode



Execution

If (current AR) ≠ 0:
    pma → PC
Else:
    (PC) + 2 → PC
Modify current AR as specified

Status Bits

None affected.

Description

The one 2-word instruction or two 1-word instructions following the branch instruction are fetched from program memory and executed before the branch is taken.

After the instructions are executed if the contents of the current auxiliary register (AR) are not 0, control is passed to the program memory address (pma); otherwise, control is passed to the next instruction. The default modification to current AR is a decrement by 1. You can cause N loop iterations to be executed by initializing the auxiliary register loop counter to N–1 before loop entry. The pma can be either a symbolic or numeric address.

BANZD is a branch and call instruction (see Table 6–8).

Words

2

Cycles

The BANZD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

Example

```
BANZD PGM0
LACC #01h
LDP #5
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	5h	AR0	4h
DP	4	DP	5
ACC	00h	ACC	01h

After the current DP and accumulator (ACC) are modified as specified, program execution continues from location 0.

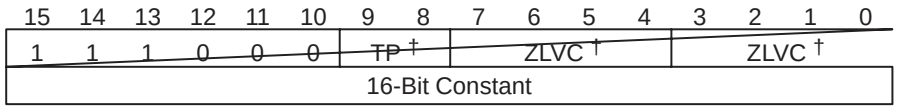
Syntax **BCND** *pma*, *cond* [,*cond1*] [,...]

Operands $0 \le pma \le 65535$

Conditions:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
TC = 0	NTC
TC = 1	TC
$\overline{BIO}$ low	BIO
Unconditionally	UNC

Opcode



† See Table 6–1 on page 6-2.

Execution

If (condition(s)):
pma → PC
Else:
(PC) + 2 → PC

Status Bits None affected.

Description If the specified conditions are met, control is passed to the program memory address (pma); otherwise, control is passed to the next instruction. Not all combinations of the conditions are meaningful and testing BIO is mutually exclusive to testing TC.

BCND is a branch and call instruction (see Table 6–8).

Words 2

Cycles The BCND instruction is not repeatable.

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+4p†
False	2	2	2	2+2p

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
BCND PGM191,LEQ,C
```

If the accumulator (ACC) contents are less than or equal to 0 and the C bit is set, program address 191 is loaded into the program counter (PC), and the program continues executing from that location. If these conditions are not met, execution continues from location PC + 2.

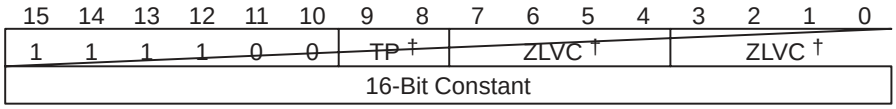
Syntax **BCNDD** *pma*, *cond* [,*cond1*] [,...]

Operands $0 \leq pma \leq 65535$

Conditions:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
TC = 0	NTC
$\overline{TC}$ = 1	TC
$\overline{BIO}$ low	BIO
Unconditionally	UNC

Opcode



† See Table 6–1 on page 6-2.

Execution

If (condition(s)):
pma → PC
Else:
(PC) + 2 → PC

Status Bits None affected.

Description The one 2-word instruction or two 1-word instructions following the branch are fetched from program memory and executed before the branch is taken. The two instruction words following the BCNDD instruction have no effect on the conditions being tested.

After the instructions are executed if the specified conditions are met, control is passed to the program memory address (pma); otherwise, control is passed to the next instruction. Not all combinations of the conditions are meaningful and testing BIO is mutually exclusive to testing TC.

BCNDD is a branch and call instruction (see Table 6–8).

Words 2

Cycles The BCNDD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

Example

```
BCNDD PGM191,OV
MAR *,AR1
LDP #5
```

After the current AR, ARP, and DP are modified as specified, program execution continues at location 191 if the overflow (OV) bit is set. If the OV bit is cleared, execution continues at the instruction following the LDP instruction.

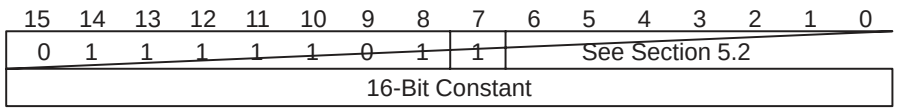
Syntax

BD *pma* [, {*ind*} [,**AR***n*]]

Operands

$0 \le pma \le 65535$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode



Execution

pma $\rightarrow$ PC
Modify current AR and ARP as specified

Status Bits

None affected.

Description

The one 2-word instruction or two 1-word instructions following the branch instruction are fetched from program memory and executed before the branch is taken.

After the instructions are executed, control is passed to the program memory address (pma). The current auxiliary register (AR) and auxiliary register pointer (ARP) are modified as specified. The pma can be either a symbolic or numeric address.

BD is a branch and call instruction (see Table 6–8).

Words

2

Cycles

The BD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

Example

```
BD 191
MAR *+,AR1
LDP #5
```

After the current AR, ARP, and DP are modified as specified, program execution continues from location 191.

Syntax

Direct: **BIT** *dma*, *bit code*
Indirect: **BIT** {*ind*} , *bit code* [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le$ bit code ≤ 15
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	BITX †				0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	BITX †				1	See Section 5.2						

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 → PC
(dma bit at bit address (15 – bit code)) → TC

Status Bits

Affects: TC

Description

The specified bit of the data memory address (dma) value is copied to the TC bit in ST1. The APL, BITT, CMPR, CPL, LST1, NORM, OPL, and XPL instructions also affect the TC bit. The bit code value corresponds to a specified bit of the dma, as given by the following table:

Bit		Bit Code			
(LSB)	0	1	1	1	1
	1	1	1	1	0
	2	1	1	0	1
	3	1	1	0	0
	4	1	0	1	1
	5	1	0	1	0
	6	1	0	0	1
	7	1	0	0	0
	8	0	1	1	1
	9	0	1	1	0
	10	0	1	0	1
	11	0	1	0	0
	12	0	0	1	1
	13	0	0	1	0
	14	0	0	0	1
(MSB)	15	0	0	0	0

BIT is a control instruction (see Table 6–10).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
BIT 0h,15 ;(DP = 6).Test LSB at 300h
```

	Before Instruction		After Instruction
Data Memory 300h	4DC8h	Data Memory 300h	4DC8h
TC	0	TC	0

Example 2

```
BIT *,0,AR1 ;Test MSB at 310h
```

	Before Instruction		After Instruction
ARP	0	ARP	1
AR0	310h	AR0	310h
Data Memory 310h	8000h	Data Memory 310h	8000h
TC	0	TC	1

Syntax

Direct: **BITT** *dma*
Indirect: **BITT** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma bit at bit address (15 –TREG2(3–0))) → TC

Status Bits

Affects: TC

Description

The specified bit of the data memory address (dma) value is copied to the TC bit in ST1. The APL, BIT, CMPR, CPL, LST1, OPL, NORM, and XPL instructions also affect the TC bit. The bit code value contained in the 4 LSBs of the TREG2 corresponds to a specified bit of the dma, as given by the following table:

Bit		Bit Code			
(LSB)	0	1	1	1	1
	1	1	1	1	0
	2	1	1	0	1
	3	1	1	0	0
	4	1	0	1	1
	5	1	0	1	0
	6	1	0	0	1
	7	1	0	0	0
	8	0	1	1	1
	9	0	1	1	0
	10	0	1	0	1
	11	0	1	0	0
	12	0	0	1	1
	13	0	0	1	0
	14	0	0	0	1
(MSB)	15	0	0	0	0

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instructions that load TREG0 to write to all three TREGs. Subsequent calls to the BITT instruction will use the TREG2 value (which is the same as TREG0), maintaining ’C5x object-code compatibility with the ’C2x.

BITT is a control instruction (see Table 6–10).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
BITT 00h ;(DP = 6). Test bit 14 of data at 300h
```

	Before Instruction		After Instruction
Data Memory 300h	4DC8h	Data Memory 300h	4DC8h
TREG2	1h	TREG2	1h
TC	0	TC	1

Example 2

```
BITT * ;Test bit 1 of data at 310h
```

	Before Instruction		After Instruction
ARP	1	ARP	1
AR1	310h	AR1	310h
Data Memory 310h	8000h	Data Memory 310h	8000h
TREG2	0Eh	TREG2	0Eh
TC	0	TC	0

Syntax

General syntax: **BLDD** *src*, *dst*

All valid cases have the general syntax:

Direct BMAR/DMA: **BLDD BMAR**, *dma*
Indirect BMAR/DMA: **BLDD BMAR**, {*ind*} [,**AR***n*]
Direct DMA/BMAR: **BLDD** *dma*, **BMAR**
Indirect DMA/BMAR: **BLDD** {*ind*}, **BMAR** [,**AR***n*]
Direct K/DMA: **BLDD** #*addr*, *dma*
Indirect K/DMA: **BLDD** #*addr*, {*ind*} [,**AR***n*]
Direct DMA/K: **BLDD** *dma*, #*addr*
Indirect DMA/K: **BLDD** {*ind*}, #*addr* [,**AR***n*]

Operands

$0 \le addr \le 65535$
$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with SRC specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	0	0	dma						

Indirect addressing with SRC specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	0	1	See Section 5.2						

Direct addressing with DEST specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	0	dma						

Indirect addressing with DEST specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	1	See Section 5.2						

Direct addressing with SRC specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	0	0	dma						
16-Bit Constant															

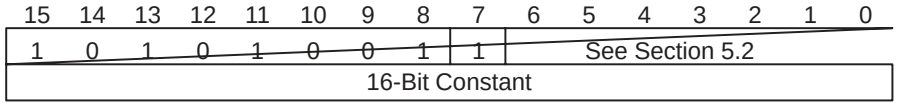
Indirect addressing with SRC specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	0	1	See Section 5.2						
16-Bit Constant															

Direct addressing with DEST specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	0	dma						
16-Bit Constant															

Indirect addressing with DEST specified by long immediate



Execution

(PFC) → MCS
If long immediate:
    (PC) + 2 → PC
    #lk → PFC
Else:
    (PC) + 1 → PC
    (BMAR) → PFC

While (repeat counter) ≠ 0:
(src, addressed by PFC) → dst or src → (dst, addressed by PFC)
Modify current AR and ARP as specified
(PFC) + 1 → PFC
(repeat counter) –1 → repeat counter
(src, addressed by PFC) → dst or src → (dst, addressed by PFC)
Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

None affected.

Description

The contents of the data memory address (dma) pointed at by *src* (source) are copied to the dma pointed at by *dst* (destination). The source and/or destination space can be pointed at by a long immediate value, the contents of the block move address register (BMAR), or a dma. Not all *src*/*dst* combinations of pointer types are valid. The source and destination blocks do not have to be entirely on-chip or off-chip.

In the indirect addressing mode, you can use the RPT instruction with the BLDD instruction to move consecutive words in data memory. The number of words to be moved is one greater than the number contained in the repeat counter register (RPTC) at the beginning of the instruction. If a long immediate value or the contents of the BMAR is specified in the repeat mode, the source and/or destination address is automatically incremented. If a dma is specified in the repeat mode, the dma address is not automatically incremented. When used with the RPT instruction, the BLDD instruction becomes a single-cycle instruction, once the RPT pipeline is started. Interrupts are inhibited during a BLDD operation used with the RPT instruction.

BLDD is an I/O and data memory operation instruction (see Table 6–9).

CAUTION

Neither the long immediate value nor the BMAR can be used as the address to the on-chip memory-mapped registers. The direct or indirect addressing mode can be used as the address to the on-chip memory-mapped registers.

Words

1 (One source or destination is specified by BMAR)

2 (One source or destination is specified by long immediate)

Cycles

Cycles for a Single Instruction (SRC or DEST in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	2	2	2	2+p
Source: SARAM Destination: DARAM	2	2	2	2+p
Source: External Destination: DARAM	2+d_{src}	2+d_{src}	2+d_{src}	2+d_{src}+p
Source: DARAM Destination: SARAM	2	2	2, 3†	2+p
Source: SARAM Destination: SARAM	2	2	2, 3†	2+p
Source: External Destination: SARAM	2+d_{src}	2+d_{src}	2+d_{src}, 3+d_{src}†	2+d_{src}+p
Source: DARAM Destination: External	3+d_{dst}	3+d_{dst}	3+d_{dst}	5+d_{dst}+p
Source: SARAM Destination: External	3+d_{dst}	3+d_{dst}	3+d_{dst}	5+d_{dst}+p
Source: External Destination: External	3+d_{src}+d_{dst}	3+d_{src}+d_{dst}	3+d_{src}+d_{dst}	5+d_{src}+d_{dst}+p

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC or DEST in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	n+1	n+1	n+1	n+1+p
Source: SARAM Destination: DARAM	n+1	n+1	n+1	n+1+p
Source: External Destination: DARAM	n+1+nd_{src}	n+1+nd_{src}	n+1+nd_{src}	n+1+nd_{src}+p
Source: DARAM Destination: SARAM	n+1	n+1	n+1, n+3†	n+1+p
Source: SARAM Destination: SARAM	n+1, 2n–1‡	n+1, 2n–1‡	n+1, 2n–1‡, n+3§, 2n+1§	n+1+p, 2n–1+p‡
Source: External Destination: SARAM	n+1+nd_{src}†	n+1+nd_{src}	n+1+nd_{src}, n+3+nd_{src}†	n+1+nd_{src}+p
Source: DARAM Destination: External	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}+p
Source: SARAM Destination: External	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}+p
Source: External Destination: External	4n–1+nd_{src} +nd_{dst}	4n–1+nd_{src} +nd_{dst}	4n–1+nd_{src} +nd_{dst}	4n+1+nd_{src}+nd_{dst}+p

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Cycles for a Single Instruction (SRC or DEST long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	3	3	3	3+2p
Source: SARAM Destination: DARAM	3	3	3	3+2p
Source: External Destination: DARAM	3+d_{src}	3+d_{src}	3+d_{src}	3+d_{src}+2p
Source: DARAM Destination: SARAM	3	3	3, 4†	3+2p
Source: SARAM Destination: SARAM	3	3	3, 4†	3+2p

† If the destination operand and the code are in the same SARAM block

Cycles for a Single Instruction (SRC or DEST long immediate) (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: External Destination: SARAM	3+d_{src}	3+d_{src}	3+d_{src}, 4+d_{src}	3+d_{src}+2p
Source: DARAM Destination: External	4+d_{dst}	4+d_{dst}	4+d_{dst}	6+d_{dst}+2p
Source: SARAM Destination: External	4+d_{dst}	4+d_{dst}	4+d_{dst}	6+d_{dst}+2p
Source: External Destination: External	4+d_{src}+d_{dst}	4+d_{src}+d_{dst}	4+d_{src}+d_{dst}	6+d_{src}+d_{dst}+2p
Source: DARAM Destination: DARAM	n+2	n+2	n+2	n+2+2p

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC or DEST long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: SARAM Destination: DARAM	n+2	n+2	n+2	n+2+2p
Source: External Destination: DARAM	n+2+nd_{src}	n+2+nd_{src}	n+2+nd_{src}	n+2+nd_{src}
Source: DARAM Destination: SARAM	n+2	n+2	n+2, n+4†	n+2+2p
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, n+4†, 2n+2§	n+2+2p, 2n+2p‡
Source: External Destination: SARAM	n+2nd_{src}	n+2nd_{src}	n+2nd_{src}, n+4+nd_{src}†	n+2+nd_{src}+2p
Source: DARAM Destination: External	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst} +2p
Source: SARAM Destination: External	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}+2p
Source: External Destination: External	4n+nd_{src}+nd_{dst}‡	4n+nd_{src}+nd_{dst}	4n+nd_{src}+nd_{dst}	4n+2+nd_{src}+nd_{dst}+2p

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Example 1

```
BLDD #300h,20h ;(DP = 6)
```

	Before Instruction	After Instruction
Data Memory 300h	0h	0h
320h	0Fh	0h

Example 2

```
BLDD *+,#321h,AR3
```

	Before Instruction	After Instruction
ARP	2	3
AR2	301h	302h
Data Memory 301h	01h	01h
321h	0Fh	01h

Example 3

```
BLDD BMAR,*
```

	Before Instruction	After Instruction
ARP	2	2
BMAR	320h	320h
AR2	340h	340h
Data Memory 320h	01h	01h
340h	0Fh	01h

Example 4

```
BLDD 00h,BMAR ;(DP = 6)
```

	Before Instruction	After Instruction
Data Memory 300h	0Fh	0Fh
BMAR	320h	320h
Data Memory 320h	01h	0Fh

Example 5

```
RPT 2
BLDD #300h,*+
```

	Before Instruction	After Instruction
ARP	0	0
AR0	320h	323h
300h	7F98h	7F98h
301h	FFE6h	FFE6h
302h	9522h	9522h
320h	8DEEh	7F98h
321h	9315h	0FFE6h
322h	2531h	9522h

Syntax

Direct: **BLDP** *dma*
Indirect: **BLDP** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(PFC) → MCS
(BMAR) → PFC

While (repeat counter) ≠ 0:
    dma → (dst, addressed by PFC)
    Modify current AR and ARP as specified
    (PFC) + 1 → PFC
    (repeat counter) –1 → repeat counter
dma → (dst, addressed by PFC)
Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

None affected.

Description

The contents of the data memory address (dma) are copied to the program memory address (pma) pointed at by the block move address register (BMAR). The source and destination blocks do not have to be entirely on-chip or off-chip.

In the indirect addressing mode, you can use the RPT instruction with the BLDP instruction to move consecutive words in data memory to a contiguous program memory space pointed at by the BMAR. The number of words to be moved is one greater than the number contained in the repeat counter register (RPTC) at the beginning of the instruction. The contents of the BMAR are automatically incremented when used in the repeat mode. When used with the RPT instruction, the BLDP instruction becomes a single-cycle instruction, once the RPT pipeline is started. Interrupts are inhibited during a BLDP operation used with the RPT instruction.

BLDP is an I/O and data memory operation instruction (see Table 6–9).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	2	2	2	2+p
Source: SARAM Destination: DARAM	2	2, 3¶	2	2+p
Source: External Destination: DARAM	2+d_{src}	2+d_{src}	2+d_{src}	3+d_{src}+p_{code}
Source: DARAM Destination: SARAM	2	2	2, 3†	2+p
Source: SARAM Destination: SARAM	2	2	2, 3† ¶, 4§	2+p
Source: External Destination: SARAM	2+d_{src}	2+d_{src}	2+d_{src}, 3+d_{src}†	3+d_{src}+p_{code}
Source: DARAM Destination: External	3+p_{dst}	3+p_{dst}	3+p_{dst}	4+p_{dst}+p_{code}
Source: SARAM Destination: External	3+p_{dst}	3+p_{dst}	3+p_{dst}, 4+p_{dst}¶	4+p_{dst}+p_{code}
Source: External Destination: External	3+d_{src}+p_{dst}	3+d_{src}+p_{dst}	3+d_{src}+p_{dst}	5+d_{src}+p_{dst}+p_{code}

† If the destination operand and the code are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	n+1	n+1	n+1	n+1+p_{code}
Source: SARAM Destination: DARAM	n+1	n+1	n+1, n+2¶	n+1+p_{code}
Source: External Destination: DARAM	n+1+nd_{src}	n+1+nd_{src}	n+1+nd_{src}	n+2+nd_{src}+p_{code}

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: SARAM	n+1	n+1	n+1, n+2†	n+1+p_{code}
Source: SARAM Destination: SARAM	n+1, 2n–1‡	n+1, 2n–1‡	n+1, 2n–1‡, n+2† ¶, 2n+1§	n+1+p_{code}, 2n–1+p_{code}‡
Source: External Destination: SARAM	n+1+nd_{src}	n+1+nd_{src}	n+1+nd_{src}, n+2+np_{src}†	n+2+nd_{src}+p_{code}
Source: DARAM Destination: External	2n+1+np_{dst}	2n+1+np_{dst}	2n+1+np_{dst}	2n+2+np_{dst}+p_{code}
Source: SARAM Destination: External	2n+1+np_{dst}	2n+1+np_{dst}	2n+1+np_{dst}, 2n+2+np_{dst}¶	2n+2+np_{dst}+p_{code}
Source: External Destination: External	4n–1+nd_{src} +np_{dst}	4n–1+nd_{sr} +np_{dst}	4n–1+nd_{src} +np_{dst}	4n+1+nd_{src}+np_{dst} +p_{code}

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ If the source operand and the code are in the same SARAM block

Example 1

```
BLDP 00h ;(DP=6)
```

	Before Instruction		After Instruction
Data Memory 300h	A089h	Data Memory 300h	A089h
BMAR	2800h	BMAR	2800h
Program Memory 2800h	1234h	Program Memory 2800h	A089h

Example 2

```
BLDP *,AR0
```

	Before Instruction		After Instruction
ARP	7	ARP	0
AR7	310h	AR7	310h
Data Memory 310h	F0F0h	Data Memory 310h	F0F0h
BMAR	2800h	BMAR	2800h
Program Memory 2800h	1234h	Program Memory 2800h	F0F0h

Syntax

General syntax: **BLPD** *src*, *dst*

All valid cases have the general syntax:

Direct BMAR/DMA: **BLPD BMAR**, *dma*

Indirect BMAR/DMA: **BLPD BMAR**, {*ind*} [,**AR***n*]

Direct K/DMA: **BLPD** #*pma*, *dma*

Indirect K/DMA: **BLPD** #*pma*, {*ind*} [,**AR***n*]

Operands

$0 \leq pma \leq 65535$

$0 \leq dma \leq 127$

$0 \leq n \leq 7$

ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with SRC specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	0	0	dma						

Indirect addressing with SRC specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	0	1	See Section 5.2						

Direct addressing with SRC specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	0	dma						
16-Bit Constant															

Indirect addressing with SRC specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	1	See Section 5.2						
16-Bit Constant															

Execution

If long immediate:

(PC) + 2 → PC

(PFC) → MCS

lk → PFC

Else:

(PC) + 1 → PC

(PFC) → MCS

(BMAR) → PFC

While (repeat counter) ≠ 0:
(pma, addressed by PFC) → dst
Modify current AR and ARP as specified
(PFC) + 1 → PFC
(repeat counter) –1 → repeat counter

(pma, addressed by PFC) → dst
Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

None affected.

Description

The contents of the program memory address (pma) pointed at by *src* (source) are copied to the data memory address (dma) pointed at by *dst* (destination). The source space can be pointed at by a long immediate value or the contents of the block move address register (BMAR). The destination space can be pointed at by a dma or the contents of current AR. Not all *src*/*dst* combinations of pointer types are valid. The source and destination blocks do not have to be entirely on-chip or off-chip.

In the indirect addressing mode, you can use the RPT instruction with the BLPD instruction to move consecutive words in program memory to data memory. The number of words to be moved is one greater than the number contained in the repeat counter register (RPTC) at the beginning of the instruction. If a long immediate value or the contents of the BMAR is specified in the repeat mode, the source address is automatically incremented. When used with the RPT instruction, the BLPD instruction becomes a single-cycle instruction, once the RPT pipeline is started. Interrupts are inhibited during a BLPD operation used with the RPT instruction.

BLPD is an I/O and data memory operation instruction (see Table 6–9).

Words

1 (Source is specified by BMAR)

2 (Source is specified by long immediate)

Cycles

Cycles for a Single Instruction (SRC in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	2	2	2	$2+p_{code}$
Source: SARAM Destination: DARAM	2	2	2	$2+p_{code}$
Source: External Destination: DARAM	$2+p_{src}$	$2+p_{src}$	$2+p_{src}$	$2+p_{src}+p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Single Instruction (SRC in BMAR) (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: SARAM	2	2	2, 3†	$2+p_{code}$
Source: SARAM Destination: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External Destination: SARAM	$2+p_{src}$	$2+p_{src}$	$2+p_{src}$, $3+p_{src}$†	$2+p_{src}+2p_{code}$
Source: DARAM/ROM Destination: External	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+p_{code}$
Source: SARAM Destination: External	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+p_{code}$
Source: External Destination: External	$3+p_{src}+d_{dst}$	$3+p_{src}+d_{dst}$	$3+p_{src}+d_{dst}$	$5+p_{src}+d_{dst}+p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	n+1	n+1	n+1	n+1+p_{code}
Source: SARAM Destination: DARAM	n+1	n+1	n+1	n+1+p_{code}
Source: External Destination: DARAM	n+1+np_{src}	n+1+np_{src}	n+1+np_{src}	n+1+np_{src}+p_{code}
Source: DARAM/ROM Destination: SARAM	n+1	n+1	n+1, n+3†	n+1+p_{code}
Source: SARAM Destination: SARAM	n+1, 2n–1‡	n+1, 2n–1‡	n+1, 2n–1‡, n+3†, 2n+1§	n+1+p_{code}, 2n–1+p_{code}‡
Source: External Destination: SARAM	n+1+np_{src}	n+1+np_{src}	n+1+np_{src}, n+3+np_{src}†	n+1+np_{src}+p_{code}
Source: DARAM/ROM Destination: External	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}+p_{code}
Source: SARAM Destination: External	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}+p_{code}
Source: External Destination: External	4n–1+np_{src} +nd_{dst}	4n–1+np_{src} +nd_{dst}	4n–1+np_{src} +nd_{dst}	4n+1+np_{src}+nd_{dst} +p_{code}

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Cycles for a Single Instruction (SRC long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	3	3	3	3+2p_{code}
Source: SARAM Destination: DARAM	3	3	3	3+2p_{code}
Source: External Destination: DARAM	3+p_{src}	3+p_{src}	3+p_{src}	3+p_{src}+2p_{code}
Source: DARAM/ROM Destination: SARAM	3	3	3, 4†	3+2p_{code}
Source: SARAM Destination: SARAM	3	3	3, 4†	3+2p_{code}

† If the destination operand and the code are in the same SARAM block

Cycles for a Single Instruction (SRC long immediate) (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: External Destination: SARAM	$3+p_{src}$	$3+p_{src}$	$3+p_{src}$, $4+p_{src}$ †	$3+p_{src}+2p_{code}$
Source: DARAM/ROM Destination: External	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+2p_{code}$
Source: SARAM Destination: External	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+2p_{code}$
Source: External Destination: External	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$6+p_{src}+d_{dst}+2p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Source: SARAM Destination: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Source: External Destination: DARAM	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}+2p_{code}$
Source: DARAM/ROM Destination: SARAM	n+2	n+2	n+2, n+4†	$n+2+2p_{code}$
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, n+4†, 2n+2§	$n+2+2p_{code}$, $2n+2p_{code}$‡
Source: External Destination: SARAM	$n+2+np_{src}$ †	$n+2+np_{src}$	$n+2+np_{src}$, $n+4+np_{src}$ †	$n+2+np_{src}+2p_{code}$
Source: DARAM/ROM Destination: External	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}+2p_{code}$
Source: SARAM Destination: External	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}+2p_{code}$
Source: External Destination: External	$4n+np_{src}+nd_{dst}$ †	$4n+np_{src}+nd_{dst}$	$4n+np_{src}+nd_{dst}$	$4n+2+np_{src}+nd_{dst}$ $+2p_{code}$

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Example 1

```
BLPD #800h,00h ;(DP=6)
```

	Before Instruction		After Instruction
Program Memory 800h	0Fh	Program Memory 800h	0Fh
Data Memory 300h	0h	Data Memory 300h	0Fh

Example 2

```
BLPD #800h,*,AR7
```

	Before Instruction		After Instruction
ARP	0	ARP	7
AR0	310h	AR0	310h
Program Memory 800h	1111h	Program Memory 800h	1111h
Data Memory 310h	0100h	Data Memory 310h	1111h

Example 3

```
BLPD BMAR,00h ;(DP=6)
```

	Before Instruction		After Instruction
BMAR	800h	BMAR	800h
Program Memory 800h	0Fh	Program Memory 800h	0Fh
Data Memory 300h	0h	Data Memory 300h	0Fh

Example 4

```
BLPD BMAR,*+,AR7
```

	Before Instruction		After Instruction
ARP	0	ARP	7
AR0	300h	AR0	301h
BMAR	810h	BMAR	810h
Program Memory 810h	4444h	Program Memory 810h	4444h
Data Memory 300h	0100h	Data Memory 300h	4444h

Syntax **BSAR** *shift*

Operands $1 \le \text{shift} \le 16$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	SHFT †			

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 → PC
(ACC) / 2^{shift} → ACC

Status Bits *Affected by:* SXM

Description The contents of the accumulator (ACC) are right-barrel arithmetic shifted 1 to 16 bits, as defined by the shift code, in a single cycle. If the SXM bit is cleared, the high-order bits of the ACC are zero-filled; if the SXM bit is set, the high-order bits of the ACC are sign-extended.

BSAR is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1

```
BSAR 16 ;(SXM=0)
```

	Before Instruction		After Instruction
ACC	0001 0000h	ACC	0000 0001h

Example 2

```
BSAR 4 ;(SXM=1)
```

	Before Instruction		After Instruction
ACC	FFF1 0000h	ACC	FFFF 1000h

Syntax **CALA**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0

Execution

(PC) + 1 → TOS
(ACC(15–0)) → PC

Status Bits None affected.

Description The current program counter (PC) is incremented and pushed onto the top of the stack (TOS). The contents of the accumulator low byte (ACCL) are loaded into the PC. Execution continues at this address.

The CALA instruction is used to perform computed subroutine calls. CALA is a branch and call instruction (see Table 6–8).

Words 1

Cycles The CALA instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
CALA
```

	Before Instruction		After Instruction
PC	25h	PC	83h
ACC	83h	ACC	83h
TOS	100h	TOS	26h

Syntax **CALAD**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	1	0	1

Execution

(PC) + 3 $\rightarrow$ TOS
(ACC(15–0)) $\rightarrow$ PC

Status Bits None affected.

Description The current program counter (PC) is incremented by 3 and pushed onto the top of the stack (TOS).

Then, the one 2-word instruction or two 1-word instructions following the CALAD instruction are fetched from program memory and executed before the call is executed.

Then, the contents of the accumulator low byte (ACCL) are loaded into the PC. Execution continues at this address.

The CALAD instruction is used to perform computed subroutine calls. CALAD is a branch and call instruction (see Table 6–8).

Words 1

Cycles The CALAD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

Example

```
CALAD
MAR *+,AR1
LDP #5
```

	Before Instruction		After Instruction
ARP	0	ARP	1
AR0	8	AR0	9
DP	0	DP	5
PC	25h	PC	83h
ACC	83h	ACC	83h
TOS	100h	TOS	28h

After the current AR, ARP, and DP are modified as specified, the address of the instruction following the LDP instruction is pushed onto the stack, and program execution continues from location 83h.

Syntax

CALL *pma* [,{*ind*} [,**AR***n*]]

Operands

0 ≤ pma ≤ 65535
0≤ n ≤ 7
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0	1	See Section 5.2						
16-Bit Constant															

Execution

(PC) + 2 → TOS
pma → PC
Modify current AR and ARP as specified

Status Bits

None affected.

Description

The current program counter (PC) is incremented and pushed onto the top of the stack (TOS). The program memory address (pma) is loaded into the PC. Execution continues at this address. The current auxiliary register (AR) and auxiliary register pointer (ARP) are modified as specified. The pma can be either a symbolic or numeric address.

CALL is a branch and call instruction (see Table 6–8).

Words

2

Cycles

The CALL instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+4p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
CALL PRG191,*+,AR0
```

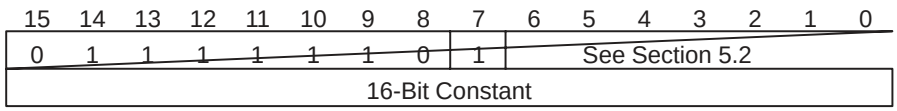
	Before Instruction		After Instruction
ARP	1	ARP	0
AR1	05h	AR1	06h
PC	30h	PC	0BFh
TOS	100h	TOS	32h

0BFh is loaded into the PC, and the program continues executing from that location.

Syntax **CALLD** *pma* [,{*ind*} [,**AR***n*]]

Operands
$0 \le pma \le 65535$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode



Execution
(PC) + 4 → TOS
pma → PC
Modify current AR and ARP as specified

Status Bits None affected.

Description The current program counter (PC) is incremented by 4 and pushed onto the top of the stack (TOS).

Then, the one 2-word instruction or two 1-word instructions following the CALLD instruction are fetched from program memory and executed before the call is executed.

Then, the program memory address (pma) is loaded into the PC. Execution continues at this address. The current auxiliary register (AR) and auxiliary register pointer (ARP) are modified as specified. The pma can be either a symbolic or numeric address.

CALLD is a branch and call instruction (see Table 6–8).

Words 2

Cycles The CALLD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

Example

```
CALLD PRG191
MAR *+,AR1
LDP #5
```

	Before Instruction		After Instruction
ARP	0	ARP	1
AR0	09h	AR0	0Ah
DP	1	DP	5
PC	30h	PC	0BFh
TOS	100h	TOS	34h

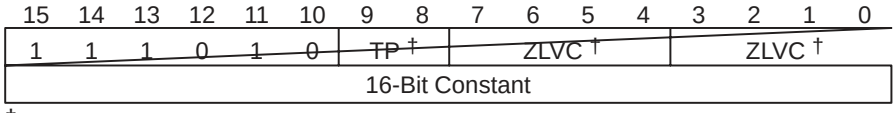
After the current AR, ARP, and DP are modified as specified, the address of the instruction following the LDP instruction is pushed onto the stack, and program execution continues from location 0BFh.

Syntax **CC** *pma cond* [,*cond1*] [,...]

Operands $0 \leq \text{pma} \leq 65535$

Conditions:		
	ACC = 0	EQ
	ACC ≠ 0	NEQ
	ACC < 0	LT
	ACC ≤ 0	LEQ
	ACC > 0	GT
	ACC ≥ 0	GEQ
	C = 0	NC
	C = 1	C
	OV = 0	NOV
	OV = 1	OV
	TC = 0	NTC
	TC = 1	TC
	$\overline{\text{BIO}}$ low	BIO
	Unconditionally	UNC

Opcode



† See Table 6–1 on page 6-2.

Execution

If (condition(s)):
(PC) + 2 → TOS
pma → PC
Else:
(PC) + 2 → PC

Status Bits None affected.

Description If the specified conditions are met, the current program counter (PC) is incremented and pushed onto the top of the stack (TOS). The program memory address (pma) is loaded into the PC. Execution continues at this address. The pma can be either a symbolic or numeric address. Not all combinations of the conditions are meaningful. In addition, the NTC, TC, and BIO conditions are mutually exclusive. If the specified conditions are not met, control is passed to the next instruction.

The CC instruction functions in the same manner as the CALL instruction (page 6-85) if all conditions are true. CC is a branch and call instruction (see Table 6–8).

Words 2

Cycles

The CC instruction is not repeatable.

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+4p†
False	2	2	2	2+2p

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
CC  PGM191,LEQ,C
```

If the accumulator (ACC) contents are less than or equal to 0 and the C bit is set, 0BFh is loaded into the program counter (PC), and the program continues executing from that location. If the conditions are not met, execution continues at the instruction following the CC instruction.

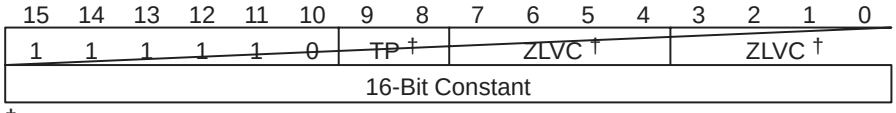
Syntax **CCD** *pma cond* [,*cond1*] [,...]

Operands $0 \leq pma \leq 65535$

Conditions:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
TC = 0	NTC
TC = 1	TC
$\overline{\text{BIO}}$ low	BIO
Unconditionally	UNC

Opcode



† See Table 6–1 on page 6-2.

Execution

If (condition(s)):
(PC) + 4 → TOS
pma → PC
Else:
(PC) + 2 → PC

Status Bits None affected.

Description If the specified conditions are met, the current program counter (PC) is incremented by 4 and pushed onto the top of the stack (TOS).

Then, the one 2-word instruction or two 1-word instructions following the CCD instruction are fetched from program memory and executed before the call is executed.

Then, the program memory address (pma) is loaded into the PC. Execution continues at this address. The pma can be either a symbolic or numeric address. Not all combinations of the conditions are meaningful. In addition, the NTC, TC, and BIO conditions are mutually exclusive.

If the specified conditions are not met, control is passed to the next instruction.

The CCD functions in the same manner as the CALLD instruction (page 6-86) if all conditions are true. CCD is a branch and call instruction (see Table 6–8).

Words 2

Cycles The CCD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

Example

```
CCD PGM191,LEQ,C
MAR *+,AR1
LDP #5
```

The current AR, ARP, and DP are modified as specified. If the accumulator (ACC) contents are less than or equal to 0 and the C bit is set, the address of the instruction following the LDP instruction is pushed onto the stack and program execution continues from location 0BFh. If the conditions are not met, execution continues at the instruction following the LDP instruction.

Syntax **CLRC** *control bit*

Operands *control bit*: {C, CNF, HM, INTM, OVM, SXM, TC, XF}

Opcode **CLRC OVM** (Clear overflow mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0

CLRC SXM (Clear sign extension mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	1	0

CLRC HM (Clear hold mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	0	0

CLRC TC (Clear test/control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	1	0

CLRC C (Clear carry)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	0

CLRC CNF (Clear configuration control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	0	0

CLRC INTM (Clear interrupt mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0

CLRC XF (Clear external flag pin)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	0	0

Execution (PC) + 1 $\rightarrow$ PC
0 $\rightarrow$ control bit

Status Bits Affects selected control bit.

Description The specified control bit is cleared. The LST instruction can also be used to load ST0 and ST1. See Section 4.4, *Status and Control Registers*, for more information on each control bit.

CLRC is a control instruction (see Table 6–10).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
CLRC TC ;TC is bit 11 of ST1
```

	Before Instruction		After Instruction
ST1	x9xxh	ST1	x1xxh

Syntax **CMPL**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1

Execution

$(PC) + 1 \rightarrow PC$
$(\overline{ACC}) \rightarrow ACC$

Status Bits *Does not affect:* C

Description The contents of the accumulator (ACC) are replaced with its logical inversion (1s complement).

CMPL is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

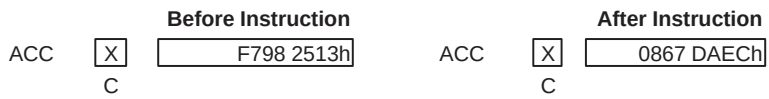
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
CMPL
```



Syntax **CMPR** *CM*

Operands $0 \leq CM \leq 3$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	1	CM †	

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 → PC
Compare (current AR) to (ARCR)
If condition true:
    1 → TC
If condition false:
    0 → TC

Status Bits

Affected by:	*Not affected by:*	*Affects*:	*Does not affect:*
NDX	SXM	TC	SXM

Description

The contents of the current auxiliary register (AR) are compared with the contents of the auxiliary register compare register (ARCR), as defined by the value of CM:

If CM = 00, test for (current AR) = (ARCR)
If CM = 01, test for (current AR) < (ARCR)
If CM = 10, test for (current AR) > (ARCR)
If CM = 11, test for (current AR) ≠ (ARCR)

If the condition is true, the TC bit is set. If the condition is false, the TC bit is cleared.

The ARs are treated as unsigned integers in the comparisons. You can maintain software compatibility with the ’C2x by clearing the NDX bit. This causes any ’C2x instruction that loads auxiliary register 0 (AR0) to load the ARCR and index register (INDX) also, maintaining ’C5x object-code compatibility with the ’C2x.

CMPR is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
CMPR 2
```

	Before Instruction		After Instruction
ARP	4	ARP	4
ARCR	FFFFh	ARCR	FFFFh
AR4	7FFFh	AR4	7FFFh
TC	1	TC	0

Syntax

Direct: **CPL** [,#*lk*] *dma*
Indirect: **CPL** [,#*lk*] {*ind*} [,**AR***n*]

Operands

0 ≤ dma ≤ 127
lk: 16-bit constant
0 ≤ n ≤ 7
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	1	0	dma						

Indirect addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	1	1	See Section 5.2						

Direct addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	0	dma						
16-Bit Constant															

Indirect addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	1	See Section 5.2						
16-Bit Constant															

Execution

Long immediate not specified:
(PC) + 1 → PC
Compare (DBMR) to (dma)
If (DBMR) = (dma):
    1 → TC
Else:
    0 → TC

Long immediate specified:
(PC) + 2 → PC
Compare lk to (dma)
If lk = (dma):
    1 → TC
Else:
    0 → TC

Status Bits

Not affected by:	*Affects*:
SXM	TC

Description

If a long immediate constant is specified, the constant is compared with the contents of the data memory address (dma). If a constant is not specified, the contents of the dma are compared with the contents of the dynamic bit manipulation register (DBMR). If the two quantities involved in the comparison are equal, the TC bit is set. If the condition is false, the TC bit is cleared.

CPL is a parallel logic unit (PLU) instruction (see Table 6–6).

Words

1 (Long immediate not specified)

2 (Long immediate specified)

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2, 3†	2+2p
External	2+d	2+d	2+d	3+d+2p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	n+1	n+1	n+1, n+2†	n+1+2p
External	n+1	n+1	n+1	n+2+2p

† If the operand and the code are in the same SARAM block

Example 1

```
CPL #060h,60h
```

	Before Instruction		After Instruction
Data Memory 60h	066h	Data Memory 60h	066h
TC	1	TC	0

Example 2

```
CPL 60h
```

	Before Instruction		After Instruction
Data Memory 60h	066h	Data Memory 60h	066h
DBMR	066h	DBMR	066h
TC	0	TC	1

Example 3

```
CPL #0F1h,*,AR6
```

	Before Instruction		After Instruction
ARP	7	ARP	6
AR7	300h	AR7	300h
Data Memory 300h	0F1h	Data Memory 300h	0F1h
TC	1	TC	1

Example 4

```
CPL *,AR7
```

	Before Instruction		After Instruction
ARP	6	ARP	7
AR6	300h	AR6	300h
Data Memory 300h	0F1h	Data Memory 300h	0F1h
DBMR	0F0h	DBMR	0F0h
TC	0	TC	0

Syntax **CRGT**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1

Execution

(PC) + 1 → PC
Compare (ACC) to (ACCB)

If (ACC) > (ACCB):
    (ACC) → ACCB
    1 → C

If (ACC) < (ACCB):
    (ACCB) → ACC
    0 → C

If (ACC) = (ACCB):
    1 → C

Status Bits *Affects:* C

Description The contents of the accumulator (ACC) are compared to the contents of the accumulator buffer (ACCB). The larger value (signed) is loaded into both registers. If the contents of the ACC are greater than or equal to the contents of the ACCB, the C bit is set; otherwise, the C bit is cleared.

CRGT is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1 CRGT

	Before Instruction		After Instruction
ACCB	4h	ACCB	5h
ACC	5h	ACC	5h
C	0	C	1

Example 2 CRGT

	Before Instruction		After Instruction
ACCB	5h	ACCB	5h
ACC	5h	ACC	5h
C	0	C	1

Syntax **CRLT**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	0	0

Execution

(PC) + 1 → PC
Compare (ACC) to (ACCB)

If (ACC) < (ACCB):
(ACC) → ACCB
1 → C

If (ACC) > (ACCB):
(ACCB) → ACC
0 → C

If (ACC) = (ACCB):
0 → C

Status Bits *Affects:* C

Description The contents of the accumulator (ACC) are compared to the contents of the accumulator buffer (ACCB). The smaller (signed) value is loaded into both registers. If the contents of the ACC are less than the contents of the ACCB, the C bit is set; otherwise, the C bit is cleared.

CRLT is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1 CRLT

	Before Instruction		After Instruction
ACCB	5h	ACCB	4h
ACC	4h	ACC	4h
C	0	C	1

Example 2 CRLT

	Before Instruction		After Instruction
ACCB	4h	ACCB	4h
ACC	4h	ACC	4h
C	1	C	0

Syntax

Direct: **DMOV** *dma*
Indirect: **DMOV** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(dma) $\rightarrow$ dma + 1

Status Bits

Affected by: CNF and OVLY

Description

The contents of the data memory address (dma) are copied to the next higher dma. The DMOV instruction works only within on-chip data RAM blocks and within any configurable RAM block that is configured as data memory. In addition, the DMOV instruction is continuous across on-chip dual-access RAM block B0 and B1 boundaries. The DMOV instruction cannot be used on external data memory or memory-mapped registers. If the DMOV instruction is used on external memory or memory-mapped registers, the DMOV instruction will read the specified memory location but will perform no operations.

When data is copied from the addressed location to the next higher location, the contents of the addressed location remain unaffected.

You can use the DMOV instruction in implementing the z^{-1} delay encountered in digital signal processing. The DMOV function is included in the LTD, MACD, and MADD instructions (see their individual descriptions on page 6-142, 6-153, and 6-158, respectively, for more information).

DMOV is an I/O and data memory operation instruction (see Table 6–9).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n–2	2n–2	2n–2, 2n+1†	2n–2+p
External	4n–2+2nd	4n–2+2nd	4n–2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
DMOV DAT8 ;(DP = 6)
```

	Before Instruction		After Instruction
Data Memory 308h	43h	Data Memory 308h	43h
Data Memory 309h	2h	Data Memory 309h	43h

Example 2

```
DMOV *,AR1
```

	Before Instruction		After Instruction
ARP	0	ARP	1
AR1	30Ah	AR1	30Ah
Data Memory 30Ah	40h	Data Memory 30Ah	40h
Data Memory 30Bh	41h	Data Memory 30Bh	40h

Syntax **EXAR**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	0	1

Execution

(PC) + 1 $\rightarrow$ PC
(ACCB) $\leftrightarrow$ (ACC)

Status Bits None affected.

Description The contents of the accumulator (ACC) are exchanged (switched) with the contents of the accumulator buffer (ACCB).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example EXAR

	Before Instruction		After Instruction
ACC	043h	ACC	02h
ACCB	02h	ACCB	043h

Syntax **IDLE**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	1	0

Execution (PC) + 1 → PC

Status Bits *Affected by:* INTM

Description The program being executed is forced to wait until an unmasked (external or internal) interrupt or reset occurs. The program counter (PC) is incremented only once, and the device remains in idle mode until interrupted.

The idle mode is exited by an unmasked interrupt, even if the INTM bit is set. If the INTM bit is set, the program continues executing at the instruction following the IDLE. If the INTM bit is cleared, the program branches to the corresponding interrupt service routine (ISR).

When an interrupt causes IDLE to be exited with the interrupts disabled (INTM = 1), no interrupt flag register (IFR) bits are cleared. The IFR bits are cleared only if interrupts are enabled and IDLE is exited by entering the ISR.

Executing the IDLE instruction causes the ’C5x to enter the power-down mode. During the idle mode, the timer and serial port peripherals are still active. Therefore, timer and peripheral interrupts, as well as reset or external interrupts, will remove the processor from the idle mode.

IDLE is a control instruction (see Table 6–10).

Words 1

Cycles The IDLE instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Example

```
IDLE ;The processor idles until a reset or unmasked
     ;interrupt occurs.
```

Syntax **IDLE2**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	1	1

Execution (PC) + 1 $\rightarrow$ PC

Status Bits *Affected by:* INTM

Description The program being executed is forced to wait until an unmasked (external or internal) interrupt or reset occurs. The functional clock input is removed from the internal device to make an extremely low-power mode possible. The program counter (PC) is incremented only once, and the device remains in idle mode until interrupted.

The idle2 mode is exited by an unmasked interrupt, even if the INTM bit is set. If the INTM bit is set, the program continues executing at the instruction following the IDLE2. If the INTM bit is cleared, then the program branches to the corresponding interrupt service routine (ISR).

When an interrupt causes IDLE2 to be exited with the interrupts disabled (INTM = 1), no interrupt flag register (IFR) bits are cleared. The IFR bits are cleared only if interrupts are enabled and IDLE2 is exited by entering the ISR.

Executing the IDLE2 instruction causes the ’C5x to enter the power-down mode. During the idle2 mode, the timer and serial port peripherals are not active. The idle2 mode is exited by a low logic level on an external interrupt ($\overline{\text{INT1}}$–$\overline{\text{INT4}}$), $\overline{\text{RS}}$, or $\overline{\text{NMI}}$ with a duration of at least five machine cycles because interrupts are not latched as in normal device operation.

IDLE2 is a control instruction (see Table 6–10).

Words 1

Cycles The IDLE2 instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Example

```
IDLE2 ;The processor idles until a reset or unmasked
      ;external interrupt occurs.
```

Syntax

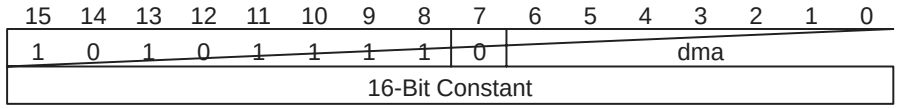
Direct: **IN** *dma*, *PA*
Indirect: **IN** {*ind*} ,*PA* [,**AR***n*]

Operands

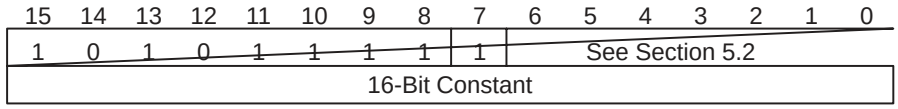
$0 \le$ dma ≤ 127
$0 \le$ n ≤ 7
$0 \le$ port address PA ≤ 65535
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indirect addressing



Execution

(PC) + 2 $\rightarrow$ PC
While (repeat counter) $\neq 0$
Port address $\rightarrow$ address bus A15–A0
Data bus D15–D0 $\rightarrow$ dma
Port address $\rightarrow$ dma
Port address + 1 $\rightarrow$ Port address
(repeat counter – 1) $\rightarrow$ repeat counter

Status Bits

None affected.

Description

A 16-bit value from an external I/O port is read into the data memory address (dma). The $\overline{\text{IS}}$ line goes low to indicate an I/O access, and the $\overline{\text{STRB}}$, $\overline{\text{RD}}$, and READY timings are the same as for an external data memory read. While port addresses 50h–5Fh are memory-mapped (see subsection 9.1.1, *Memory-Mapped Peripheral Registers and I/O Ports*); the other port addresses are not.

You can use the RPT instruction with the IN instruction to read consecutive words in I/O space to data space. The number of words to be moved is one greater than the number contained in the repeat counter register (RPTC) at the beginning of the instruction. When used with the RPT instruction, the IN instruction becomes a single-cycle instruction, once the RPT pipeline is started, and the port address is incremented after each access.

IN is an I/O and data memory operation instruction (see Table 6–9).

Words

2

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM	$2+io_{src}$	$2+io_{src}$	$2+io_{src}$	$3+io_{src}+2p_{code}$
Destination: SARAM	$2+io_{src}$	$2+io_{src}$	$2+io_{src}$, $3+io_{src}$ †	$3+io_{src}+2p_{code}$
Destination: External	$3+d_{dst}+io_{src}$	$3+d_{dst}+io_{src}$	$3+d_{dst}+io_{src}$	$6+d_{dst}+io_{src}+2p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$	$2n+1+nio_{src}+2p_{code}$
Destination: SARAM	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$, $2n+2+nio_{src}$ †	$2n+1+nio_{src}+2p_{code}$
Destination: External	$4n-1+nd_{dst}+nio_{src}$	$4n-1+nd_{dst}+nio_{src}$	$4n-1+nd_{dst}+nio_{src}$	$4n+2+nd_{dst}+nio_{src}+2p_{code}$

† If the destination operand and the code are in the same SARAM block

Example 1

```
IN DAT7,PA5 ;Read in word from peripheral on port
            ;address 5(i.e., I/O port 55h). Store in
            ;data memory location 307h (DP=6).
```

Example 2

```
IN *,1024 ;Read in word from peripheral on I/O
          ;port 400h. Store in data memory location
          ;specified by current auxiliary register.
```

Syntax **INTR** *K*

Operands $0 \le K \le 31$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	1	INTR# †				

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 → stack
corresponding *interrupt* vector → PC

Status Bits

Not affected by:	*Affects*:
INTM	INTM

Description

A software interrupt that transfers program control to a program memory address (pma) interrupt vector specified by *K*. The current program counter (PC) is incremented and pushed onto the stack. The pma is loaded into the PC. The *K* value corresponds to a pma specified by the following table:

K	Interrupt	Hex Location	K	Interrupt	Hex Location
0	$\overline{\text{RS}}$	0	16	Reserved	20
1	$\overline{\text{INT1}}$	2	17	TRAP	22
2	$\overline{\text{INT2}}$	4	18	$\overline{\text{NMI}}$	24
3	$\overline{\text{INT3}}$	6	19	Reserved	26
4	TINT	8	20	User-defined	28
5	RINT	A	21	User-defined	2A
6	XINT	C	22	User-defined	2C
7	TRNT	E	23	User-defined	2E
8	TXNT	10	24	User-defined	30
9	$\overline{\text{INT4}}$	12	25	User-defined	32
10	Reserved	14	26	User-defined	34
11	Reserved	16	27	User-defined	36
12	Reserved	18	28	User-defined	38
13	Reserved	1A	29	User-defined	3A
14	Reserved	1C	30	User-defined	3C
15	Reserved	1E	31	User-defined	3E

The INTR instruction allows any interrupt service routine (ISR) to be executed from your software. The INTM bit has no affect on the INTR instruction. An INTR interrupt for the $\overline{\text{INT1}}$–$\overline{\text{INT4}}$ interrupts looks exactly like an external interrupt except the interrupt will not clear the appropriate bit in the IFR. See Section 4.8, *Interrupts*, on page 4-36 for a complete description of interrupt operation.

INTR is a branch and call instruction (see Table 6–8).

The reserved interrupt vectors can be used for the ’C50, ’C51, and ’C53. However, software compatibility with other fifth generation devices is not guaranteed.

Words 1

Cycles The INTR instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The ’C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
INTR 3 ;Control is passed to program memory location 6h
       ;PC + 1 is pushed onto the stack.
```

Syntax **LACB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1

Execution

(PC) + 1 → PC
(ACCB) → ACC

Status Bits None affected.

Description The contents of the accumulator buffer (ACCB) are loaded into the accumulator (ACC).

LACB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example LACB

	Before Instruction		After Instruction
ACC	01376h	ACC	5555 AAAAh
ACCB	5555 AAAAh	ACCB	5555 AAAAh

Syntax

Direct: **LACC** *dma* [,*shift*]
Indirect: **LACC** {*ind*} [,*shift* [,**AR***n*]]
Long immediate: **LACC** #*lk* [,*shift*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le shift \le 16$ (defaults to 0)
$-32768 \le lk \le 32767$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	SHFT †				0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	SHFT †				1	See Section 5.2						

† See Table 6–1 on page 6-2.

Direct addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	dma						

Indirect addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	1	See Section 5.2						

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	0	0	SHFT †			
16-Bit Constant															

† See Table 6–1 on page 6-2.

Execution

Direct or indirect addressing:
(PC) + 1 $\rightarrow$ PC
(dma) $\times$ 2^{shift1} $\rightarrow$ ACC

Long immediate addressing:
(PC) + 2 $\rightarrow$ PC
lk $\times$ 2^{shift2} $\rightarrow$ ACC

Status Bits

Affected by: SXM

Description

The contents of the data memory address (dma) or a 16-bit constant are shifted left, as defined by the shift code, and loaded into the accumulator (ACC). During shifting, the low-order bits of the ACC are zero-filled. If the SXM bit is cleared, the high-order bits of the ACC are zero-filled; if the SXM bit is set, the high-order bits of the ACC are sign-extended.

LACC is an accumulator memory reference instruction (see Table 6–4).

Words

1 (Direct or indirect addressing)

2 (Long immediate addressing)

Cycles

For the long immediate addressing modes, the LACC instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

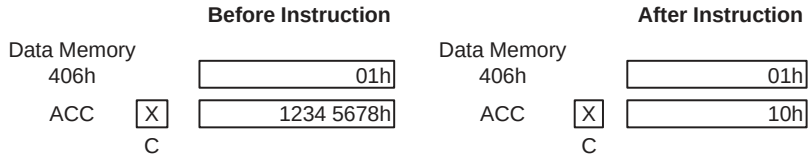
† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

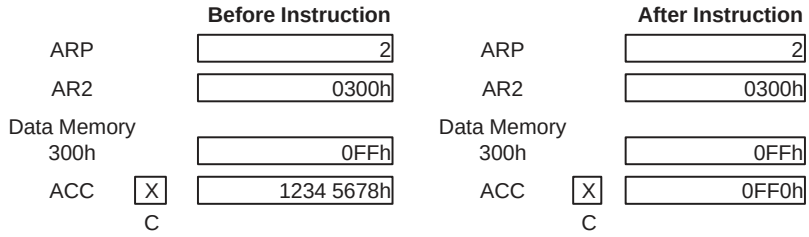
Example 1

```
LACC DAT6,4 ;(DP = 8, SXM = 0)
```



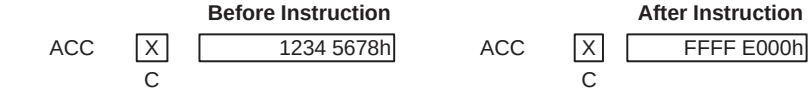
Example 2

```
LACC *,4 ;(SXM = 0)
```



Example 3

```
LACC #F000h,1 ;(SXM = 1)
```



Syntax

Direct: **LACL** *dma*
Indirect: **LACL** {*ind*} [,**AR***n*]
Short immediate: **LACL** #*k*

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le k \le 255$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	1	8-Bit Constant							

Execution

(PC) + 1 $\rightarrow$ PC

Direct or indirect addressing:
0 $\rightarrow$ ACC(31–16)
(dma) $\rightarrow$ ACC(15–0)

Short immediate addressing:
0 $\rightarrow$ ACC(31–8)
k $\rightarrow$ ACC(7–0)

Status Bits

Not affected by: SXM

Description

The contents of the data memory address (dma) or a zero-extended 8-bit constant are loaded into the accumulator low byte (ACCL). The accumulator high byte (ACCH) is zero-filled. The data is treated as an unsigned 16-bit number rather than a 2s-complement number. The operand is not sign extended with the LACL instruction, regardless of the state of the SXM bit.

LACL is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

For the short immediate addressing modes, the LACL instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

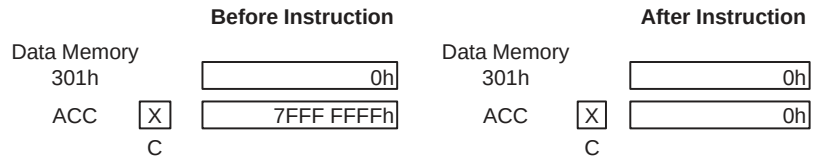
† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

	ROM	DARAM	SARAM	External Memory
	1	1	1	1+p

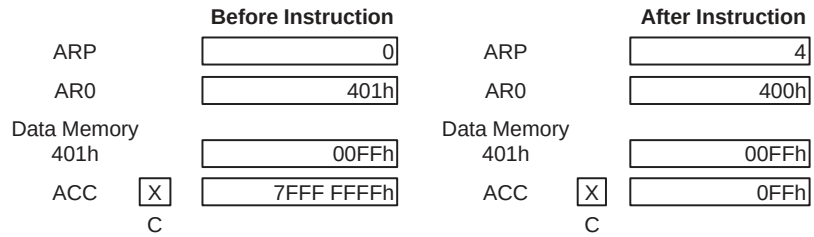
Example 1

```
LACL DAT1 ;(DP = 6)
```

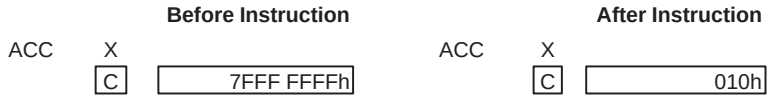


Example 2

```
LACL *-,AR4
```



Example 3 `LACL #10h`



Syntax

Direct: **LACT** *dma*
Indirect: **LACT** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(dma) $\times$ $2^{TREG1(3–0)}$ $\rightarrow$ ACC

If SXM = 0:
(dma) is not sign extended
If SXM = 1:
(dma) is sign extended

Status Bits

Affected by: SXM

Description

The contents of the data memory address (dma) are shifted left from 0 to 15 bits, as defined by the 4 LSBs of TREG1, and loaded into the accumulator (ACC). You can use the contents of TREG1 as a shift code to provide a dynamic shift mechanism. During shifting, if the SXM bit is cleared, the high-order bits are zero-filled; if the SXM bit is set, the high-order bits are sign-extended.

You may use the LACT instruction to denormalize a floating-point number, if the actual exponent is placed in the 4 LSBs of the TREG1 and the mantissa is referenced by the dma. You can use this method of denormalization only when the magnitude of the exponent is 4 bits or less.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs. Subsequent calls to the LACT instruction will shift the value by the TREG1 value (which is the same as TREG0), maintaining ’C5x object-code compatibility with the ’C2x.

LACT is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
LACT DAT1 ;(DP = 6, SXM = 0)
```

	Before Instruction	After Instruction
Data Memory 301h	1376h	1376h
ACC	98F7 EC83h	13760h
X		
C		
TREG1	14h	14h

Example 2

```
LACT *–,AR3 ;(SXM = 1)
```

	Before Instruction	After Instruction
ARP	1	3
AR1	310h	309h
Data Memory 310h	FF00h	FF00h
ACC	98F7 EC83h	FFFF FE00h
X		
C		
TREG1	11h	11h

Syntax

Direct: **LAMM** *dma*
Indirect: **LAMM** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	Data Memory Address						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → ACC(15–0)
0 → ACC(31–16)

Status Bits

Not affected by: SXM

Description

The contents of the addressed memory-mapped register are loaded into the accumulator low byte (ACCL). The accumulator high byte (ACCH) is zero-filled. The 9 MSBs of the data memory address are cleared, regardless of the current value of data memory page pointer (DP) bits or the upper 9 bits of the current AR. The LAMM instruction allows any memory location on data memory page 0 to be loaded into the ACC without modifying the DP bits.

LAMM is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
MMR†	1	1	1	1+p
MMPORT	$1+io_{src}$	$1+io_{src}$	$1+iod_{src}$	$1+2+p+iod_{src}$

† Add one more cycle for peripheral memory-mapped access

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
MMR‡	n	n	n	n+p
MMPORT	$n+mio_{src}$	$n+mio_{src}$	$n+mio_{src}$	$n+p+mio_{src}$

‡ Add *n* more cycles for peripheral memory-mapped access

Example 1

```
LAMM BMAR ;(DP = 6)
```

	Before Instruction		After Instruction
ACC	2222 1376h	ACC	5555h
BMAR	5555h	BMAR	5555h
Data Memory 31Fh	1000h	Data Memory 31Fh	1000h

Example 2

```
LAMM *
```

	Before Instruction		After Instruction
ARP	1	ARP	1
AR1	325h	AR1	325h
ACC	2222 1376h	ACC	0Fh
PRD	0Fh	PRD	0Fh
Data Memory 325h	1000h	Data Memory 325h	1000h

The value in data memory location 325h is not loaded into the ACC, the value at data memory location 25h (address of the PRD) is loaded into the ACC.

Syntax

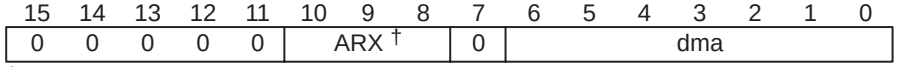
Direct: **LAR AR***x*, *dma*
Indirect: **LAR AR***x*, {*ind*} [,**AR***n*]
Short immediate: **LAR AR***x*, #*k*
Long immediate: **LAR AR***x*, #*lk*

Operands

$0 \le x \le 7$
$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le k \le 255$
$0 \le lk \le 65535$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

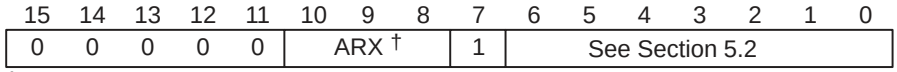
Opcode

Direct addressing



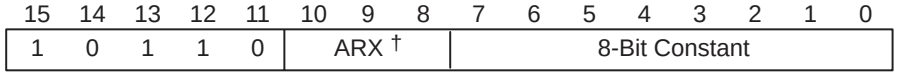
† See Table 6–1 on page 6-2.

Indirect addressing



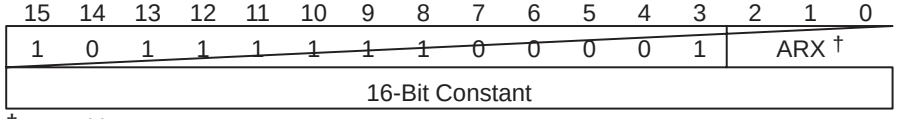
† See Table 6–1 on page 6-2.

Short immediate addressing



† See Table 6–1 on page 6-2.

Long immediate addressing



† See Table 6–1 on page 6-2.

Execution

Direct or indirect addressing:
(PC) + 1 → PC
(dma) → AR

Short immediate addressing:
(PC) + 1 → PC
k → AR

Long immediate addressing:
(PC) + 2 → PC
lk → AR

Status Bits

Affected by:	*Not affected by:*
NDX	SXM

Description

The contents of the data memory address (dma), an 8-bit constant, or a 16-bit constant are loaded into the auxiliary register (AR). The constant is acted upon like an unsigned integer, regardless of the value of the SXM bit.

You can maintain software compatibility with the ’C2x by clearing the NDX bit. This causes any ’C2x instruction that loads auxiliary register 0 (AR0) to load the auxiliary register compare register (ARCR) and index register (INDX) also, maintaining ’C5x object-code compatibility with the ’C2x.

You can use the LAR and SAR (store auxiliary register) instructions to load and store the ARs during subroutine calls and interrupts. If you do not use an AR for indirect addressing, LAR and SAR enable the register to be used as an additional storage register, especially for swapping values between data memory locations without affecting the contents of the accumulator (ACC).

LAR is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words

1 (Direct, indirect, or short immediate addressing)

2 (Long immediate addressing)

Cycles

For the short and long immediate addressing modes, the LAR instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2n	2n	2n	$2n+p_{code}$
Source: SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
Source: External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

Operand	ROM	DARAM	SARAM	External Memory
	2	2	2	$2+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

Example 1

```
LAR AR0,DAT16 ;(DP = 6)
```

	Before Instruction		After Instruction
Data Memory 310h	18h	Data Memory 310h	18h
AR0	6h	AR0	18h

Example 2

```
LAR AR4,*–
```

	Before Instruction		After Instruction
ARP	4	ARP	4
Data Memory 300h	32h	Data Memory 300h	32h
AR4	300h	AR4	32h

Note:

LAR in the indirect addressing mode ignores any AR modifications if the AR specified by the instruction is the same as that pointed to by the ARP. Therefore, in Example 2, AR4 is not decremented after the LAR instruction.

Example 3

```
LAR AR4,#01h
```

	Before Instruction		After Instruction
AR4	FF09h	AR4	01h

Example 4

```
LAR AR4,#3FFFh
```

	Before Instruction		After Instruction
AR4	0h	AR4	3FFFh

Syntax

Direct: **LDP** *dma*
Indirect: **LDP** {*ind*} [,**AR***n*]
Short immediate: **LDP** #*k*

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le k \le 511$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	9-Bit Constant								

Execution

(PC) + 1 $\rightarrow$ PC

Direct or indirect addressing:
Nine LSBs of (dma) $\rightarrow$ DP bits

Short immediate addressing:
k $\rightarrow$ DP bits

Status Bits

Affects: DP

Description

The 9 LSBs of the data memory address (dma) contents or a 9-bit constant are loaded into the data memory page pointer (DP) bits. The DP bits and the 7-bit dma are concatenated to form the 16-bit dma. The DP bits can also be loaded by the LST instruction.

LDP is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words

1

Cycles

For the short immediate addressing modes, the LDP instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2n	2n	2n	$2n+p_{code}$
Source: SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
Source: External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

Operand	ROM	DARAM	SARAM	External Memory
	2	2	2	$2+p_{code}$

† If the source operand and the code are in the same SARAM block

Example 1

```
LDP DAT127 ;(DP = 511)
```

	Before Instruction		After Instruction
Data Memory FFFFh	FEDCh	Data Memory FFFFh	FEDCh
DP	1FFh	DP	0DCh

Example 2

```
LDP #0h
```

	Before Instruction		After Instruction
DP	1FFh	DP	0h

Example 3 `LDP *,AR5`

	Before Instruction		After Instruction
ARP	4	ARP	5
AR4	300h	AR4	300h
Data Memory 300h	06h	Data Memory 300h	06h
DP	1FFh	DP	06h

Syntax

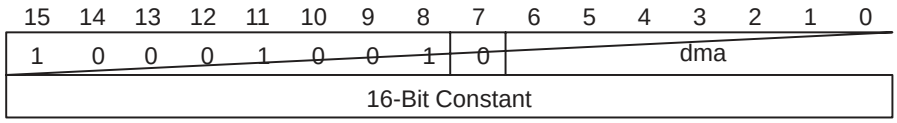
Direct: **LMMR** *dma*, *#addr*
Indirect: **LMMR** {*ind*}, *#addr* [,**AR***n*]

Operands

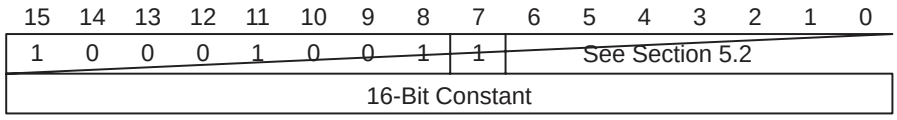
$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le addr \le 65535$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indirect addressing



Execution

PFC $\rightarrow$ MCS
(PC) + 2 $\rightarrow$ PC
lk $\rightarrow$ PFC
While (repeat counter $\neq$ 0):
    (src, addressed by PFC) $\rightarrow$ (dst, specified by lower 7 bits of dma)
    (PFC) + 1 $\rightarrow$ PFC
    (repeat counter) – 1 $\rightarrow$ repeat counter
MCS $\rightarrow$ PFC

Status Bits

None affected.

Description

The memory-mapped register pointed at by the lower 7 bits of the data memory address (dma) is loaded with the contents of the data memory location addressed by the 16-bit source address, *#addr*. The 9 MSBs of the dma are cleared, regardless of the current value of the data memory page pointer (DP) bits or the upper 9 bits of the current AR. The LMMR instruction allows any memory location on data memory page 0 to be loaded from anywhere in data memory without modification of the DP bits.

When you use the LMMR instruction with the RPT instruction, the source address, *#addr*, is incremented after every memory-mapped load operation.

LMMR is an I/O and data memory operation instruction (see Table 6–9).

Words

2

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: MMR‡	2	2	2	$2+2p_{code}$
Source: SARAM Destination: MMR‡	2	2	2, 3†	$2+2p_{code}$
Source: External Destination: MMR‡	$2+p_{src}$	$2+p_{src}$	$2+p_{src}$	$3+p_{src}+2p_{code}$
Source: DARAM Destination: MMPORT	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$	$5+2p_{code}+io_{dst}$
Source: SARAM Destination: MMPORT	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$, 4†	$5+2p_{code}+io_{dst}$
Source: External Destination: MMPORT	$3+p_{src}+io_{dst}$	$3+p_{src}+io_{dst}$	$3+p_{src}+io_{dst}$	$6+p_{src}+2p_{code}+io_{dst}$

† If the source operand and the code are in the same SARAM block
‡ Add one more cycle for peripheral memory-mapped register access

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: MMR§	2n	2n	2n	$2n+2p_{code}$
Source: SARAM Destination: MMR§	2n	2n	2n, 2n+1†	$2n+2p_{code}$
Source: External Destination: MMR§	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+2p_{code}$
Source: DARAM Destination: MMPORT	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+3+nio_{dst}+2p_{code}$
Source: SARAM Destination: MMPORT	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$, $3n+1+nio_{dst}$†	$3n+3+nio_{dst}+2p_{code}$
Source: External Destination: MMPORT	$4n-1+nd_{src}$ $+nio_{dst}$	$4n-1+nd_{src}$ $+nio_{dst}$	$4n-1+nd_{src}$ $+nio_{dst}$	$4n+2+nd_{src}+ nio_{dst}$ $+2p_{code}$

† If the source operand and the code are in the same SARAM block
§ Add *n* more cycles for peripheral memory-mapped register access

Example 1

```
LMMR DBMR,#300h
```

	Before Instruction		After Instruction
Data Memory 300h	1376h	Data Memory 300h	1376h
DBMR	5555h	DBMR	1376h

Example 2

```
LMMR *,#300h,AR4 ;CBCR = 1Eh
```

	Before Instruction		After Instruction
ARP	0	ARO	4h
AR0	31Eh	AR0	31Eh
Data Memory 300h	20h	Data Memory 300h	20h
CBCR	0h	CBCR	20h

Syntax

Direct: **LPH** *dma*
Indirect: **LPH** {*ind*} [,**AR***n*]

Operands

0 ≤ dma ≤ 127
0 ≤ n ≤ 7
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → PREG (31–16)

Status Bits

None affected.

Description

The contents of the data memory address (dma) are loaded into the product register (PREG) high byte. The contents of the PREG low byte are unaffected.

You can use the LPH instruction to restore the contents of the PREG high byte after interrupts and subroutine calls, if automatic context save is not used.

LPH is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
LPH DAT0 ;(DP = 4)
```

	Before Instruction		After Instruction
Data Memory 200h	F79Ch	Data Memory 200h	F79Ch
PREG	3007 9844h	PREG	F79C 9844h

Example 2

```
LPH *,AR6
```

	Before Instruction		After Instruction
ARP	5	ARP	6
AR5	200h	AR5	200h
Data Memory 200h	F79Ch	Data Memory 200h	F79Ch
PREG	3007 9844h	PREG	F79C 9844h

Syntax

Direct: **LST** #*m*, *dma*
Indirect: **LST** #*m*, {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$m = 0$ or 1
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing for LST #0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	0	dma						

Indirect addressing for LST #0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	1	See Section 5.2						

Direct addressing for LST #1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	dma						

Indirect addressing for LST #1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(dma) $\rightarrow$ STm
dma (13–15) $\rightarrow$ ARP (regardless of n)

Status Bits

Affects:	*Does not affect:*
ARB, ARP, C, CNF, DP, HM, OV, OVM, PM, SXM, TC, and XF	INTM

Description

The contents of the data memory address (dma) are loaded into status register STm. The INTM bit is unaffected by an LST #0 instruction. In addition, the LST #0 instruction does not affect the auxiliary register buffer (ARB), even though a new auxiliary register pointer (ARP) is loaded. If a next ARP value is specified via the indirect addressing mode, the specified value is ignored. Instead, ARP is loaded with the value contained within the addressed data memory word.

Note:

When ST1 is loaded (LST #1), the value loaded into ARB is also loaded into ARP.

You can use the LST instruction to restore the status registers after subroutine calls and interrupts. LST is a control instruction (see Table 6–10).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2n	2n	2n	$2n+p_{code}$
Source: SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
Source: External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Example 1

```
MAR *,AR0
LST #0,*,AR1 ;The data memory word addressed by the contents
             ;of auxiliary register AR0 is loaded into
             ;status register ST0,except for the INTM bit.
             ;Note that even though a next ARP value is
             ;specified, that value is ignored, and the
             ;old ARP is not loaded into the ARB.
```

Example 2

```
LST #0,60h ;(DP = 0)
```

	Before Instruction		After Instruction
Data Memory 60h	2404h	Data Memory 60h	2404h
ST0	6E00h	ST0	2604h
ST1	0580h	ST1	0580h

Example 3

```
LST #0,*–,AR1
```

	Before Instruction		After Instruction
ARP	4	ARP	7
AR4	3FFh	AR4	3FEh
Data Memory 3FFh	EE04h	Data Memory 3FFh	EE04h
ST0	1E00h	ST0	EE04h
ST1	F7A0h	ST1	F7A0h

Example 4

```
LST #1,00h ;(DP = 6)
```

	Before Instruction		After Instruction
Data Memory 300h	E1BCh	Data Memory 300h	E1BCh
ST0	0406h	ST0	E406h
ST1	09A0h	ST1	E1BCh

Syntax

Direct: **LT** *dma*
Indirect: **LT** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → TREG0

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by: TRM

Description

The contents of the data memory address (dma) are loaded into TREG0. You can use the LT instruction to load TREG0 in preparation for multiplication.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

LT is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
LT DAT24 ;(DP = 8. TRM = 1)
```

	Before Instruction		After Instruction
Data Memory 418h	62h	Data Memory 418h	62h
TREG0	3h	TREG0	62h

Example 2

```
LT *,AR3 ;(TRM = 0)
```

	Before Instruction		After Instruction
ARP	2	ARP	3
AR2	418h	AR2	418h
Data Memory 418h	62h	Data Memory 418h	62h
TREG0	3h	TREG0	62h
TREG1	4h	TREG1	2h
TREG2	5h	TREG2	2h

Syntax

Direct: **LTA** *dma*
Indirect: **LTA** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → TREG0
(ACC) + (shifted PREG) → ACC

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by: OVM, PM, and TRM

Affects: C and OV

Description

The contents of the data memory address (dma) are loaded into TREG0. The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the accumulator (ACC). The result is stored in the ACC. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

LTA is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
LTA DAT36 ;(DP = 6, PM = 0, TRM = 1)
```

	Before Instruction	After Instruction
Data Memory 324h	62h	62h
TREG0	3h	62h
PREG	0Fh	0Fh
ACC	5h	14h
C	X	0

Example 2

```
LTA *,5 ;(TRM = 0)
```

	Before Instruction	After Instruction
ARP	4	5
AR4	324h	324h
Data Memory 324h	62h	62h
TREG0	3h	62h
TREG1	4h	2h
TREG2	5h	2h
PREG	0Fh	0Fh
ACC	5h	14h
C	X	0

Syntax

Direct: **LTD** *dma*
Indirect: **LTD** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → TREG0
(dma) → dma + 1
(ACC) + (shifted PREG) → ACC

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by:	*Affects*:
OVM, PM, and TRM	C and OV

Description

The contents of the data memory address (dma) are loaded into TREG0. The contents of the dma are also copied to the next higher dma. The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the accumulator (ACC). The result is stored in the ACC. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared. See the DMOV instruction, page 6-104, for information on the data move feature.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

The LTD instruction functions in the same manner as the LTA instruction with the addition of *data move* for on-chip RAM blocks. If you use the LTD instruction with external data memory, its function is identical to that of the LTA instruction (page 6-140).

LTD is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

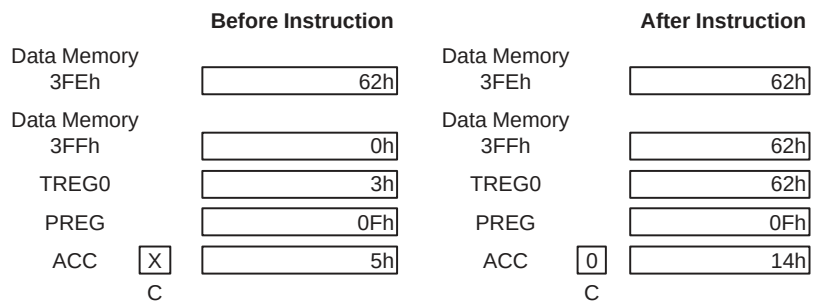
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n–2	2n–2	2n–2, 2n+1†	2n–2+p
External	4n–2+2nd	4n–2+2nd	4n–2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
LTD DAT126 ;(DP = 7, PM = 0, TRM = 1)
```



Example 2

```
LTD *,AR3 ;(TRM = 0)
```

	Before Instruction		After Instruction
ARP	1	ARP	3
AR1	3FEh	AR1	3FEh
Data Memory 3FEh	62h	Data Memory 3FEh	62h
Data Memory 3FFh	0h	Data Memory 3FFh	62h
TREG0	3h	TREG0	62h
TREG1	4h	TREG1	2h
TREG2	5h	TREG2	2h
PREG	0Fh	PREG	0Fh
ACC	5h	ACC	14h
C	X	C	0

Syntax

Direct: **LTP** *dma*
Indirect: **LTP** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → TREG0
(shifted PREG) → ACC

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by: PM and TRM

Description

The contents of the data memory address (dma) are loaded into TREG0. The contents of the product register (PREG) are shifted, as defined by the PM bits, and stored in the accumulator (ACC).

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

LTP is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
LTP DAT36 ;(DP = 6, PM = 0, TRM = 1)
```

	Before Instruction		After Instruction	
Data Memory 324h		62h	Data Memory 324h	62h
TREG0		3h	TREG0	62h
PREG		0Fh	PREG	0Fh
ACC	X	5h	ACC	X 0Fh
	C			C

Example 2

```
LTP *,AR5 ;(PM = 0, TRM = 0)
```

	Before Instruction		After Instruction	
ARP		2	ARP	5
AR2		324h	AR2	324h
Data Memory 324h		62h	Data Memory 324h	62h
TREG0		3h	TREG0	62h
TREG1		4h	TREG1	2h
TREG2		5h	TREG2	2h
PREG		0Fh	PREG	0Fh
ACC	X	5h	ACC	X 0Fh
	C			C

Syntax

Direct: **LTS** *dma*
Indirect: **LTS** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(dma) → TREG0
(ACC) – (shifted PREG) → ACC

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by:	*Affects*:
OVM, PM, and TRM	C and OV

Description

The contents of the data memory address (dma) are loaded into TREG0. The contents of the product register (PREG) are shifted, as defined by the PM bits, and subtracted from the accumulator (ACC). The result is stored in the ACC. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

LTS is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
LTS DAT36 ;(DP = 6, PM = 0, TRM = 1)
```

	Before Instruction		After Instruction	
Data Memory 324h		62h	Data Memory 324h	62h
TREG0		3h	TREG0	62h
PREG		0Fh	PREG	0Fh
ACC	X	05h	ACC	0 FFFF FFF6h
	C			C

Example 2

```
LTS *,AR2 ;(TRM = 0)
```

	Before Instruction		After Instruction	
ARP		1	ARP	2
AR1		324h	AR1	324h
324h		62h	324h	62h
TREG0		3h	TREG0	62h
TREG1		4h	TREG1	2h
TREG2		5h	TREG2	2h
PREG		0Fh	PREG	0Fh
ACC	X	05h	ACC	0 FFFF FFF6h
	C			C

Syntax

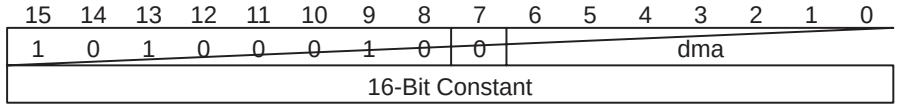
Direct: **MAC** *pma*, *dma*
Indirect: **MAC** *pma*, {*ind*} [,**AR***n*]

Operands

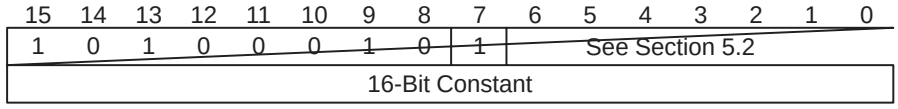
$0 \le pma \le 65535$
$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indirect addressing



Execution

(PC) + 2 → PC
(PFC) → MCS
(pma) → PFC

If (repeat counter) ≠ 0:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
Modify current AR and ARP as specified
(PFC) + 1 → PFC
(repeat counter) – 1 → repeat counter

Else:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
Modify current AR and ARP as specified
(MCS) → PFC

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by:	*Affects*:
OVM, PM, and TRM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the accumulator (ACC). The result is stored in the ACC. The contents of the data memory address (dma) are loaded into TREG0. The

contents of the dma are multiplied by the contents of the program memory address (pma). The result is stored in the PREG. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

The data and program memory locations on the ’C5x can be any nonreserved on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, then the CNF bit must be set. When the MAC instruction is used in the direct addressing mode, the dma cannot be modified during repetition of the instruction.

When the MAC instruction is repeated, the pma contained in the prefetch counter (PFC) is incremented by 1 during its operation. This allows access to a series of operands in memory. When used with the RPT instruction, the MAC instruction is useful for long sum-of-products operations because the instruction becomes a single-cycle instruction, once the RPT pipeline is started.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

MAC is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 2

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	3	3	3	$3+2p_{code}$
1: SARAM 2: DARAM	3	3	3	$3+2p_{code}$
1: External 2: DARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	3	3	3	$3+2p_{code}$
1: SARAM 2: SARAM	3, 4†	3, 4†	3, 4†	$3+2p_{code}$, $4+2p_{code}$†
1: External 2: SARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$

† If both operands are in the same SARAM block.

Cycles for a Single Instruction (Continued)

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: External	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: SARAM 2: External	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: External 2: External	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}+2p_{code}$

† If both operands are in the same SARAM block.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: SARAM 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: External 2: DARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: SARAM 2: SARAM	n+2, $2n+2^{\dagger}$	n+2, $2n+2^{\dagger}$	n+2, $2n+2^{\dagger}$	$n+2+2p_{code}$, $2n+2^{\dagger}$
1: External 2: SARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
1: DARAM/ROM 2: External	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
1: SARAM 2: External	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
1: External 2: External	$2n+2+np_{op1}$ $+nd_{op2}$	$2n+2+np_{op1}$ $+nd_{op2}$	$2n+2+np_{op1}$ $+nd_{op2}$	$2n+2+np_{op1}+nd_{op2}$ $+2p_{code}$

† If both operands are in the same SARAM block.

Example 1

```
MAC 0FF00h,02h ;(DP = 6, PM = 0, CNF = 1)
```

	Before Instruction	After Instruction
Data Memory 302h	23h	23h
Program Memory FF00h	4h	4h
TREG0	45h	23h
PREG	0045 8972h	8Ch
ACC	0723 EC41h	0769 75B3h
C	X	0

Example 2

```
MAC 0FF00h,*,AR5 ;(PM = 0, CNF = 1)
```

	Before Instruction	After Instruction
ARP	4	5
AR4	302h	302h
Data Memory 302h	23h	23h
Program Memory FF00h	4h	4h
TREG0	45h	23h
PREG	0045 8972h	8Ch
ACC	0723 EC41h	0769 75B3h
C	X	0

Syntax

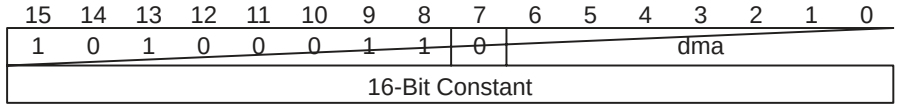
Direct: **MACD** *pma*, *dma*
Indirect: **MACD** *pma*, {*ind*} [,**AR***n*]

Operands

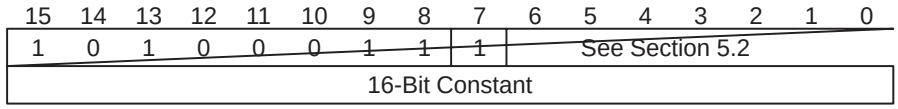
$0 \le pma \le 65535$
$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indirect addressing



Execution

(PC) + 2 → PC
(PFC) → MCS
(pma) → PFC

If (repeat counter) ≠ 0:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
Modify current AR and ARP as specified
(PFC) + 1 → PFC
(dma) → (dma) + 1
(repeat counter) – 1 → repeat counter

Else:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
(dma) → (dma) + 1
Modify current AR and ARP as specified
(MCS) → PFC

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by:	*Affects*:
OVM, PM, and TRM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the accumulator (ACC). The result is stored in the ACC. The contents of the data memory address (dma) are loaded into TREG0. The contents of the dma are multiplied by the contents of the program memory address (pma). The result is stored in the PREG. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared. See the DMOV instruction, page 6-104, for information on the data move feature.

The data and program memory locations on the ’C5x can be any nonreserved on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, then the CNF bit must be set. When the MACD instruction is used in the direct addressing mode, the dma cannot be modified during repetition of the instruction. If the MACD instruction addresses one of the memory-mapped registers or external memory as a data memory location, the effect of the instruction will be that of a MAC instruction.

When the MACD instruction is repeated, the pma contained in the prefetch counter (PFC) is incremented by 1 during its operation. This allows access to a series of operands in memory. When used with the RPT instruction, the MACD instruction becomes a single-cycle instruction, once the RPT pipeline is started.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

The MACD instruction functions in the same manner as the MAC instruction with the addition of *data move* for on-chip RAM blocks. The data move feature makes the MACD instruction useful for applications such as convolution and transversal filtering. If you use the MACD instruction with external data memory, its function is identical to that of the MAC instruction (page 6-149).

MACD is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

2

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	3	3	3	$3+2p_{code}$
1: SARAM 2: DARAM	3	3	3	$3+2p_{code}$
1: External 2: DARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	3	3	3	$3+2p_{code}$
1: SARAM 2: SARAM	3	3	3, 4‡, 5§	$3+2p_{code}$, $4+2p_{code}$‡
1: External 2: SARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
1: DARAM/ROM 2: External¶	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: SARAM 2: External¶	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: External 2: External¶	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}+2p_{code}$

‡ If both operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

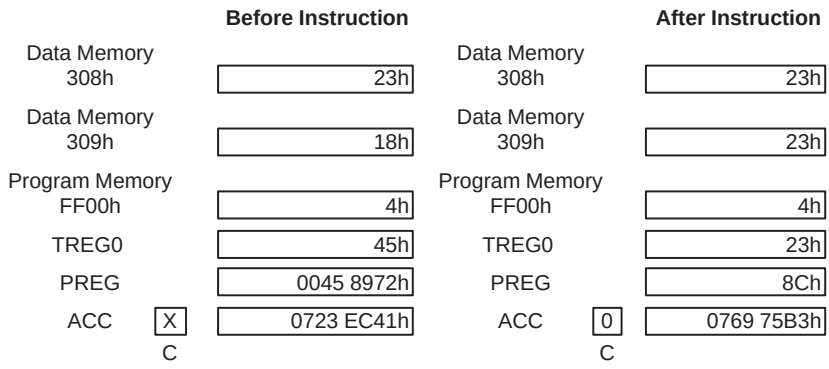
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+2	n+2	n+2	n+2+2p_{code}
1: SARAM 2: DARAM	n+2	n+2	n+2	n+2+2p_{code}
1: External 2: DARAM	n+2+np_{op1}	n+2+np_{op1}	n+2+np_{op1}	n+2+np_{op1}+2p_{code}
1: DARAM/ROM 2: SARAM	2n	2n	2n, 2n+2†	2n+2p_{code}
1: SARAM 2: SARAM	2n, 3n‡	2n, 3n‡	2n, 2n+2†, 3n‡, 3n+2§	2n+2p_{code}, 3n‡
1: External 2: SARAM	2n+np_{op1}	2n+np_{op1}	2n+np_{op1}, 2n+2+np_{op1}†	2n+np_{op1}+2p_{code}
1: DARAM/ROM 2: External¶	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}+2p_{code}
1: SARAM 2: External¶	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}+2p_{code}
1: External 2: External¶	2n+2+np_{op1} +nd_{op2}	2n+2+np_{op1} +nd_{op2}	2n+2+np_{op1} +nd_{op2}	2n+2+np_{op1}+nd_{op2} +2p_{code}

† If operand2 and code are in the same SARAM block
‡ If both operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

Example 1

```
MACD 0FF00h,08h ;(DP = 6, PM = 0, CNF = 1)
```



Example 2

```
MACD 0FF00h,*,AR6 ;(PM = 0, CF = 1)
```

	Before Instruction		After Instruction
ARP	5	ARP	6
AR5	308h	AR5	308h
Data Memory 308h	23h	Data Memory 308h	23h
Data Memory 309h	18h	Data Memory 309h	23h
Program Memory FF00h	4h	Program Memory FF00h	4h
TREG0	45h	TREG0	23h
PREG	0045 8972h	PREG	8Ch
ACC	0723 EC41h	ACC	0769 75B3h
X		0	
C		C	

The data move function for MACD can occur only within on-chip data RAM blocks.

Syntax

Direct: **MADD** *dma*
Indirect: **MADD** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	See Section 5.2						

Execution

(PC) + 2 → PC
(PFC) → MCS
(BMAR) → PFC

If (repeat counter) ≠ 0:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
Modify current AR and ARP as specified
(PFC) + 1 → PFC
(dma) → (dma) + 1
(repeat counter) – 1 → repeat counter

Else:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
(dma) → (dma) + 1
Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

Affected by:	*Affects*:
OVM and PM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the accumulator (ACC). The result is stored in the ACC. The contents of the data memory address (dma) are loaded into TREG0. The contents of the dma are multiplied by the contents of the program memory address (pma). The result is stored in the PREG. The pma is contained in the block move address register (BMAR) and is not specified by a long immediate constant; this enables dynamic addressing of coefficient tables. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared. See the DMOV instruction, page 6-104, for information on the data move feature.

The data and program memory locations on the ’C5x can be any nonreserved on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, then the CNF bit must be set. When the MADD instruction is used in the direct addressing mode, the dma cannot be modified during repetition of the instruction. If the MADD instruction addresses one of the memory-mapped registers or external memory as a data memory location, the effect of the instruction is that of a MADS instruction.

When the MADD instruction is repeated, the pma contained in the prefetch counter (PFC) is incremented by 1 during its operation. This allows access to a series of operands in memory. When used with the RPT instruction, the MADD instruction becomes a single-cycle instruction, once the RPT pipeline is started.

The MADD instruction functions in the same manner as the MADS instruction with the addition of *data move* for on-chip RAM blocks. The data move feature makes the MADD instruction useful for applications such as convolution and transversal filtering. If you use the MADD instruction with external data memory, its function is identical to that of the MADS instruction (page 6-162).

MADD is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	2	2	2	$2+p_{code}$
1: SARAM 2: DARAM	2	2	2	$2+p_{code}$
1: External 2: DARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}+p_{code}$
1: DARAM/ROM 2: SARAM	2	2	2	$2+p_{code}$
1: SARAM 2: SARAM	2	2	2, 3‡, 4§	$2+p_{code}$, $3+p_{code}$‡
1: External 2: SARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}+p_{code}$

‡ If both operands are in the same SARAM block
§ If both operands and code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

Cycles for a Single Instruction (Continued)

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: External¶	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: SARAM 2: External¶	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: External 2: External¶	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}+p_{code}$

‡ If both operands are in the same SARAM block
§ If both operands and code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: SARAM 2: DARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: External 2: DARAM	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}+p_{code}$
1: DARAM/ROM 2: SARAM	2n–1	2n–1	2n–1, 2n+1†	$2n-1+p_{code}$
1: SARAM 2: SARAM	2n–1, 3n–1‡	2n–1, 3n–1‡	2n–1, 2n+1†, 3n–1‡, 3n+1§	$2n-1+p_{code}$, 3n–1‡
1: External 2: SARAM	$2n-1+np_{op1}$	$2n-1+np_{op1}$	$2n-1+np_{op1}$, $2n+1+np_{op1}$†	$2n-1+np_{op1}+p_{code}$
1: DARAM/ROM 2: External¶	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}+p_{code}$
1: SARAM 2: External¶	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}+p_{code}$
1: External 2: External¶	$2n+1+np_{op1}$ $+nd_{op2}$	$2n+1+np_{op1}$ $+nd_{op2}$	$2n+1+np_{op1}$ $+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}$ $+p_{code}$

† If operand2 and code are in the same SARAM block
‡ If both operands are in the same SARAM block
§ If both operands and code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

Example 1

```
MADD DAT7 ;(DP = 6, PM = 0, CNF = 1)
```

	Before Instruction		After Instruction	
Data Memory 307h		8h		8h
Data Memory 308h		9h		8h
BMAR		FF00h		FF00h
TREG0		4Eh		8h
FF00h		2h		2h
PREG		0045 8972h		10h
ACC	X	0723 EC41h	0	0769 75B3h
	C		C	

Example 2

```
MADD *,3 ;(PM = 0, CNF = 1)
```

	Before Instruction		After Instruction	
ARP		2		3
AR2		307h		307h
Data Memory 307h		8h		8h
Data Memory 308h		9h		8h
BMAR		FF00h		FF00h
TREG0		4Eh		8h
FF00h		2h		2h
PREG		0045 8972h		10h
ACC	X	0723 EC41h	0	0769 75B3h
	C		C	

The data move function for MADD can occur only within on-chip data RAM blocks.

Syntax

Direct: **MADS** *dma*
Indirect: **MADS** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(PFC) → MCS
(BMAR) → PFC

If (repeat counter) ≠ 0:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
Modify current AR and ARP as specified
(PFC) + 1 → PFC
(repeat counter) – 1 → repeat counter

Else:
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (pma, addressed by PFC) → PREG
Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

Affected by:	*Affects*:
OVM and PM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the accumulator (ACC). The result is stored in the ACC. The contents of the data memory address (dma) are loaded into TREG0. The contents of the dma are multiplied by the contents of the program memory address (pma). The result is stored in the PREG. The pma is contained in the block move address register (BMAR) and is not specified by a long immediate constant; this enables dynamic addressing of coefficient tables. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

The data and program memory locations on the ’C5x can be any nonreserved on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, then the CNF bit must be set. When the MADS instruction is used in the direct addressing mode, the dma cannot be modified during repetition of the instruction.

When the MADS instruction is repeated, the pma contained in the prefetch counter (PFC) is incremented by 1 during its operation. This allows access to a series of operands in memory. When used with the RPT instruction, the MADS instruction is useful for long sum-of-products operations because the instruction becomes a single-cycle instruction, once the RPT pipeline is started.

MADS is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	2	2	2	$2+p_{code}$
1: SARAM 2: DARAM	2	2	2	$2+p_{code}$
1: External 2: DARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	
1: DARAM/ROM 2: SARAM	2	2	2	$2+p_{code}$
1: SARAM 2: SARAM	2, 3†	2, 3†	2, 3†	$2+p_{code}$, $3+p_{code}$†
1: External 2: SARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}+p_{code}$
1: DARAM/ROM 2: External	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: SARAM 2: External	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: External 2: External	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}+p_{code}$

† If both operands are in the same SARAM block.

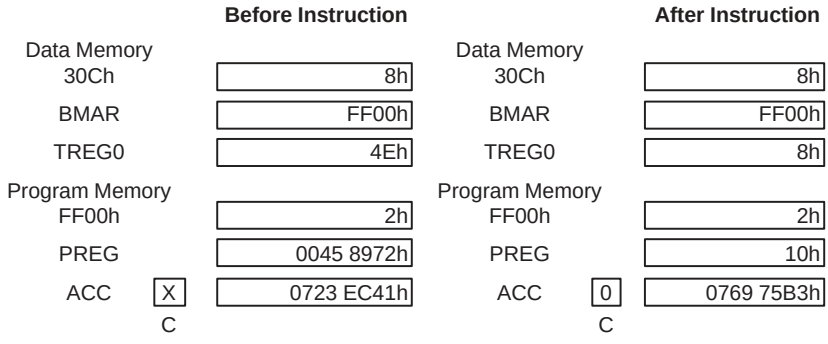
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+1	n+1	n+1	n+1+p_{code}
1: SARAM 2: DARAM	n+1	n+1	n+1	n+1+p_{code}
1: External 2: DARAM	n+1+np_{op1}	n+1+np_{op1}	n+1+np_{op1}	n+1+np_{op1}+p_{code}
1: DARAM/ROM 2: SARAM	n+1	n+1	n+1	n+1+p_{code}
1: SARAM 2: SARAM	n+1, 2n+1†	n+1, 2n+1†	n+1, 2n+1†	n+1+p_{code}, 2n+1†
1: External 2: SARAM	n+1+np_{op1}	n+1+np_{op1}	n+1+np_{op1}	n+1+np_{op1}+p_{code}
1: DARAM/ROM 2: External	n+1+nd_{op2}	n+1+nd_{op2}	n+1+nd_{op2}	n+1+nd_{op2}+p_{code}
1: SARAM 2: External	n+1+nd_{op2}	n+1+nd_{op2}	n+1+nd_{op2}	n+1+nd_{op2}+p_{code}
1: External 2: External	2n+1+np_{op1} +nd_{op2}	2n+1+np_{op1} +nd_{op2}	2n+1+np_{op1} +nd_{op2}	2n+1+np_{op1}+nd_{op2} +p_{code}

† If both operands are in the same SARAM block.

Example 1

```
MADS DAT12 ;(DP = 6, PM = 0, CNF = 1)
```



Example 2

```
MADS *,AR3 ;(PM = 0, CNF = 1)
```

	Before Instruction		After Instruction
ARP	2	ARP	3
AR2	30Ch	AR2	30Ch
Data Memory 30Ch	8h	Data Memory 30Ch	8h
BMAR	FF00h	BMAR	FF00h
TREG0	4Eh	TREG0	8h
Program Memory FF00h	2h	Program Memory FF00h	2h
PREG	0045 8972h	PREG	10h
ACC	0723 EC41h	ACC	0769 75B3h
C	X	C	0

Syntax

Direct: **MAR** *dma*
Indirect: **MAR** {*ind*} [,**AR***n*]

Operands

$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	See Section 5.2						

Execution

(PC) + 1 → PC

Indirect addressing:
Modify current AR and ARP as specified

Direct addressing:
Executes as a NOP

Status Bits

Affected by: NDX

Description

In the indirect addressing mode, the auxiliary registers (ARs) and the auxiliary register pointer (ARP) are modified; however, the memory being referenced is unaffected.

You can maintain software compatibility with the ’C2x by clearing the NDX bit. This causes any ’C2x instruction that modifies AR0 to modify the auxiliary register compare register (ARCR) and index register (INDX) also, maintaining ’C5x object-code compatibility with the ’C2x.

The MAR instruction modifies the ARs or the ARP bits, and the old ARP bits are copied to the auxiliary register buffer (ARB) bits. Any operation performed with the MAR instruction can also be performed with any instruction that supports indirect addressing. The ARP bits can also be loaded by an LST instruction.

Note:

The LARP instruction from the ’C2x instruction set is a subset of the MAR instruction (that is, MAR *,4 performs the same function as LARP 4).

MAR is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words

1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1

```
MAR *,AR1 ;Load ARP with 1
```

	Before Instruction		After Instruction
ARP	0	ARP	1
ARB	7	ARB	0

Example 2

```
MAR *+,AR5 ;Increment current auxiliary register (AR1)
           ;and load ARP with 5.
```

	Before Instruction		After Instruction
AR1	34h	AR1	35h
ARP	1	ARP	5
ARB	0	ARB	1

Syntax

Direct: **MPY** *dma*
Indirect: **MPY** {*ind*} [,**AR***n*]
Short immediate: **MPY** #*k*
Long immediate: **MPY** #*lk*

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$-4096 \le k \le 4095$
$-32768 \le lk \le 32767$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	13-Bit Constant												

Long immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0
16-Bit Constant															

Execution

Direct or indirect addressing:
(PC) + 1 → PC
(TREG0) × (dma) → PREG

Short immediate addressing:
(PC) + 1 → PC
(TREG0) × k → PREG

Long immediate addressing:
(PC) + 2 → PC
(TREG0) × lk → PREG

Status Bits

Affected by:	*Not affected by:*
TRM	SXM

Description

If a constant is specified, the constant is multiplied by the contents of TREG0. If a constant is not specified, the contents of TREG0 are multiplied by the contents of the data memory address (dma). The result is stored in the product register (PREG). Short immediate addressing multiplies TREG0 by a signed 13-bit constant. The short immediate constant is right-justified and sign-extended before the multiplication, regardless of the SXM bit.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

MPY is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1 (Direct, indirect, or short immediate addressing)

2 (Long immediate addressing)

Cycles

For the short and long immediate addressing modes, the MPY instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

	ROM	DARAM	SARAM	External Memory
	1	1	1	1+p

Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

Example 1 `MPY DAT13 ;(DP = 8)`

	Before Instruction		After Instruction
Data Memory 40Dh	7h	Data Memory 40Dh	7h
TREG0	6h	TREG0	6h
PREG	36h	PREG	2Ah

Example 2 `MPY *,AR2`

	Before Instruction		After Instruction
ARP	1	ARP	2
AR1	40Dh	AR1	40Dh
Data Memory 40Dh	7h	Data Memory 40Dh	7h
TREG0	6h	TREG0	6h
PREG	36h	PREG	2Ah

Example 3 `MPY #031h`

	Before Instruction		After Instruction
TREG0	2h	TREG0	2h
PREG	36h	PREG	62h

Example 4 `MPY #01234h`

	Before Instruction		After Instruction
TREG0	2h	TREG0	2h
PREG	36h	PREG	2468h

Syntax

Direct: **MPYA** *dma*
Indirect: **MPYA** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) + (shifted PREG) $\rightarrow$ ACC
(TREG0) $\times$ (dma) $\rightarrow$ PREG

Status Bits

Affected by:	*Affects*:
OVM, PM, and TRM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the contents of the accumulator (ACC). The result is stored in the ACC. The contents of TREG0 are multiplied by the contents of the data memory address (dma). The result is stored in the PREG. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

MPYA is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
MPYA DAT13 ;(DP = 6, PM = 0)
```

	Before Instruction		After Instruction	
Data Memory 30Dh	7h		7h	
TREG0	6h		6h	
PREG	36h		2Ah	
ACC	54h	X	8Ah	0
		C		C

Example 2

```
MPYA *,AR4 ;(PM = 0)
```

	Before Instruction		After Instruction	
ARP	3		4	
AR3	30Dh		30Dh	
Data Memory 30Dh	7h		7h	
TREG0	6h		6h	
PREG	36h		2Ah	
ACC	54h	X	8Ah	0
		C		C

Syntax

Direct: **MPYS** *dma*
Indirect: **MPYS** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) – (shifted PREG) $\rightarrow$ ACC
(TREG0) $\times$ (dma) $\rightarrow$ PREG

Status Bits

Affected by:	*Affects*:
OVM, PM, and TRM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and subtracted from the contents of the accumulator (ACC). The result is stored in the ACC. The contents of TREG0 are multiplied by the contents of the data memory address (dma). The result is stored in the PREG. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

MPYS is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

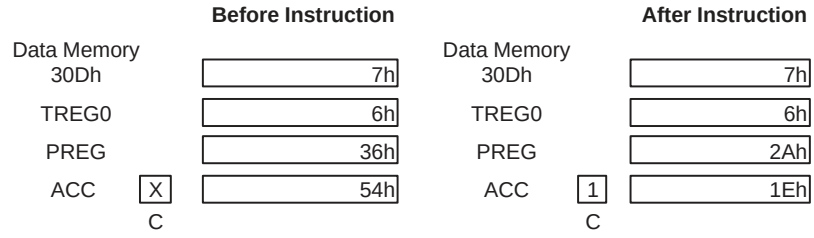
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

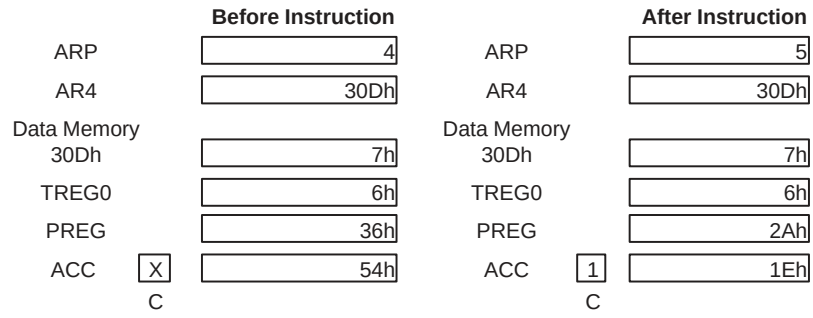
Example 1

```
MPYS DAT13 ;(DP = 6, PM = 0)
```



Example 2

```
MPYS *,AR5 ;(PM = 0)
```



Syntax

Direct: **MPYU** *dma*
Indirect: **MPYU** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
Unsigned (TREG0) $\times$ unsigned (dma) $\rightarrow$ PREG

Status Bits

Affected by:	*Not affected by*:
TRM	SXM

Description

The unsigned contents of TREG0 are multiplied by the unsigned contents of the data memory address (dma). The result is stored in the product register (PREG). The multiplier acts as a signed 17 $\times$ 17-bit multiplier for this instruction, with the MSB of both operands forced to 0.

The p-scaler shifter at the output of the PREG always invokes sign-extension on the PREG, when the PM bits are set to 11_2 (right-shift-by-6 mode). Therefore, you should not use this shift mode if you want unsigned products.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

The MPYU instruction is particularly useful for computing multiple-precision products, such as multiplying two 32-bit numbers to yield a 64-bit product. MPYU is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
MPYU DAT16 ;(DP = 4)
```

	Before Instruction		After Instruction
Data Memory 210h	FFFFh	Data Memory 210h	FFFFh
TREG0	FFFFh	TREG0	FFFFh
PREG	1h	PREG	FFFE 0001h

Example 2

```
MPYU *,AR6
```

	Before Instruction		After Instruction
ARP	5	ARP	6
AR5	210h	AR5	210h
Data Memory 210h	FFFFh	Data Memory 210h	FFFFh
TREG0	FFFFh	TREG0	FFFFh
PREG	1h	PREG	FFFE 0001h

Syntax **NEG**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0

Execution

(PC) + 1 → PC
(ACC) × –1 → ACC

If (ACC) ≠ 0:
0 → C

If (ACC) = 0:
1 → C

Status Bits

Affected by:	*Affects*:
OVM	C and OV

Description The contents of the accumulator (ACC) are replaced with its arithmetic complement (2s complement). If the contents of the ACC are not 0, the C bit is cleared; if the contents of the ACC are 0, the C bit is set.

When taking the 2s complement of 8000 0000h, the OV bit is set and: if the OVM bit is set, the ACC is replaced with 7FFF FFFFh; if the OVM bit is cleared, the ACC is replaced with 8000 0000h.

NEG is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

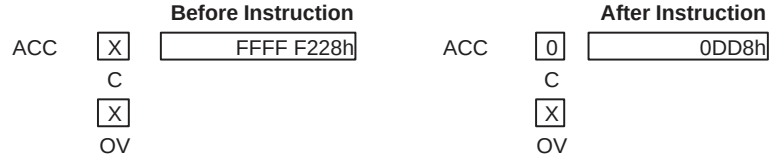
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

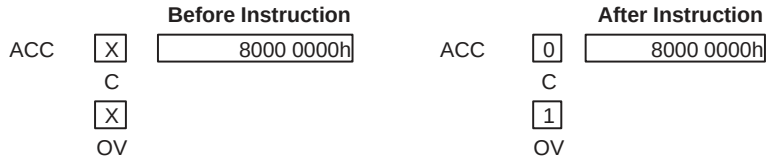
Example 1

```
NEG ;(OVM = X)
```



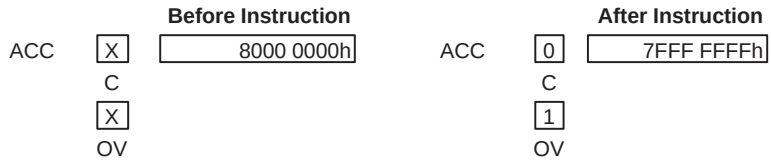
Example 2

```
NEG ;(OVM = 0)
```



Example 3

```
NEG ;(OVM = 1)
```



Syntax **NMI**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	0	0	1	0

Execution

(PC) + 1 → stack
24h → PC
1 → INTM

Status Bits

Not affected by:
INTM

Affects:
INTM

Description

The current program counter (PC) is incremented and pushed onto the stack. The nonmaskable interrupt vector located at 24h is loaded into the PC. Execution continues at this address. Interrupts are globally disabled (INTM bit is set). The NMI instruction has the same affect as a hardware nonmaskable interrupt. Automatic context save is not performed.

NMI is a branch and call instruction (see Table 6–8).

Words 1

Cycles The NMI instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The ’C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
NMI ;Control is passed to program memory location 24h
    ;and PC+1 is pushed onto the stack.
```

Syntax **NOP**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0

Execution (PC) + 1 $\rightarrow$ PC

Status Bits None affected.

Description No operation is performed. The NOP instruction affects only the program counter (PC). You can use the NOP instruction to create pipeline and execution delays.

NOP is a control instruction (see Table 6–10).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
NOP ;No operation is performed
```

Syntax **NORM** {*ind*}

Operands ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC

If (ACC) = 0:
    TC → 1

Else:
    If (ACC(31)) **XOR** (ACC(30)) = 0:
        TC → 0
        (ACC) × 2 → ACC
        Modify current AR as specified
    Else:
        TC → 1

Status Bits *Affects:* TC

Description The signed number contained in the accumulator (ACC) is normalized. Normalizing a fixed-point number separates the number into a mantissa and an exponent by finding the magnitude of the sign-extended number. ACC bit 31 is exclusive-ORed (XOR) with ACC bit 30 to determine if bit 30 is part of the magnitude or part of the sign extension. If the bits are the same, then they are both sign bits, and the ACC is shifted left to eliminate the extra sign bit. If the result of the XOR operation is true, the TC bit is set; otherwise, the TC bit is cleared.

The current AR is modified as specified to generate the magnitude of the exponent. It is assumed that the current AR is initialized before normalization begins. The default modification of the current AR is an increment.

Multiple executions of the NORM instruction may be required to completely normalize a 32-bit number in the ACC. Although using NORM with RPT does not cause execution of NORM to fall out of the repeat loop automatically when the normalization is complete, no operation is performed for the remainder of the repeat loop. The NORM instruction functions on both positive and negative 2s-complement numbers.

NORM is an accumulator memory reference instruction (see Table 6–4).

CAUTION

The NORM instruction executes the auxiliary register operation during the execution phase of the pipeline. Therefore, the auxiliary register used in the NORM instruction should not be used by an auxiliary register instruction in the next two instruction words immediately following the NORM instruction. Also, the auxiliary register pointer (ARP) should not be modified by the next two words.

Words 1

Cycles

Cycles for a Single Instruction

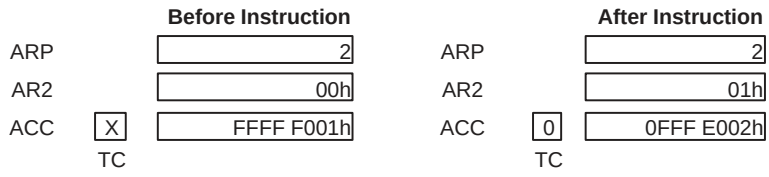
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1

```
NORM *+
```



Example 2 31-bit normalization:

```
      MAR   *,AR1     ;Use AR1 to store the exponent.
      LAR   AR1,#0h   ;Clear out exponent counter.
LOOP  NORM  *+        ;One bit is normalized.
      BCND  LOOP,NTC  ;If TC = 0, magnitude not found yet.
```

Example 3 15-bit normalization:

```
MAR*,AR1        ;Use AR1 to store the exponent.
LAR   AR1,#0Fh  ;Initialize exponent counter.
RPT   #14       ;15–bit normalization specified (yielding
                ;a 4–bit exponent and 16–bit mantissa).
NORM  *–        ;NORM automatically stops shifting when
                ;first significant magnitude bit is found,
                ;performing NOPs for the remainder of the
                ;repeat loops
```

The method in Example 2 normalizes a 32-bit number and yields a 5-bit exponent magnitude. The method in Example 3 normalizes a 16-bit number and yields a 4-bit magnitude. If the number requires only a small amount of normalization, the Example 2 method may be preferable to the Example 3 method because the loop in Example 2 runs only until normalization is complete; Example 3 always executes all 15 cycles of the repeat loop. Specifically, Example 2 is more efficient if the number requires three or fewer shifts. If the number requires six or more shifts, Example 3 is more efficient.

Note:

The NORM instruction can be used without a specified operand. In that case, any comments on the same line as the instruction are interpreted as the operand. If the first character is an asterisk (*), then the instruction is assembled as NORM * with no auxiliary register modification taking place upon execution. Therefore, TI recommends that you replace the NORM instructions with NORM *+ when you want the default increment modification.

Syntax

Direct: **OPL** [#*lk*], *dma*
Indirect: **OPL** [#*lk*], {*ind*} [,**AR***n*]

Operands

0 ≤ dma ≤ 127
lk: 16-bit constant
0 ≤ n ≤ 7
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	0	dma						

Indirect addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	1	See Section 5.2						

Direct addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	1	0	dma						
16-Bit Constant															

Indirect addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	1	1	See Section 5.2						
16-Bit Constant															

Execution

Long immediate not specified:
(PC) + 1 → PC
(dma) **OR** (DBMR) → dma

Long immediate specified:
(PC) +2 → PC
(dma) **OR** lk → dma

Status Bits

Affects: TC

Description

If a long immediate constant is specified, the constant is ORed with the contents of the data memory address (dma). If a constant is not specified, the contents of the dma are ORed with the contents of the dynamic bit manipulation register (DBMR). In both cases, the result is written directly back to the dma and the contents of the accumulator (ACC) are unaffected. If the result of the OR operation is 0, the TC bit is set; otherwise, the TC bit is cleared.

OPL is a parallel logic unit (PLU) instruction (see Table 6–6).

Words

1 (Long immediate not specified)

2 (Long immediate specified)

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n–2	2n–2	2n–2, 2n+1†	2n–2+p
External	4n–2+2nd	4n–2+2nd	4n–2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2	2+2p
External	3+2d	3+2d	3+2d	6+2d+2p

Cycles for a Repeat (RPT) Execution (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	2n–1	2n–1	2n–1, 2n+2†	2n–1+2p
External	4n–1+2nd	4n–1+2nd	4n–1+2nd	4n+2+2nd+2p

† If the operand and the code are in the same SARAM block

Example 1

```
OPL DAT10 ;(DP=6)
```

	Before Instruction		After Instruction
DBMR	FFF0h	DBMR	FFF0h
Data Memory 30Ah	0001h	Data Memory 30Ah	FFF1h

Example 2

```
OPL #0FFFh,DAT10 ;(DP=6)
```

	Before Instruction		After Instruction
Data Memory 30Ah	0001h	Data Memory 30Ah	0FFFh

Example 3

```
OPL *,AR6
```

	Before Instruction		After Instruction
ARP	3	ARP	6
AR3	300h	AR3	300h
DBMR	00F0h	DBMR	00F0h
Data Memory 300h	000Fh	Data Memory 300h	00FFh

Example 4

```
OPL #1111h,*,AR3
```

	Before Instruction		After Instruction
ARP	6	ARP	3
AR6	306h	AR6	306h
Data Memory 306h	0Eh	Data Memory 306h	111Fh

Syntax

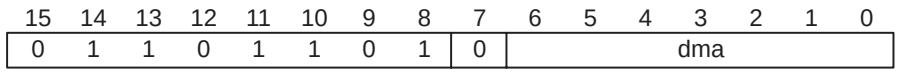
Direct: **OR** *dma*
Indirect: **OR** {*ind*} [,**AR***n*]
Long immediate: **OR** #*lk* [,*shift*]

Operands

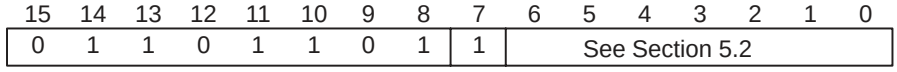
$0 \le dma \le 127$
$0 \le n \le 7$
lk: 16-bit constant
$0 \le shift \le 16$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

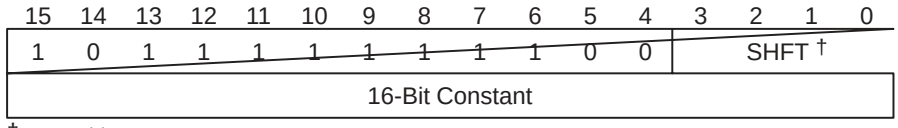
Direct addressing



Indirect addressing

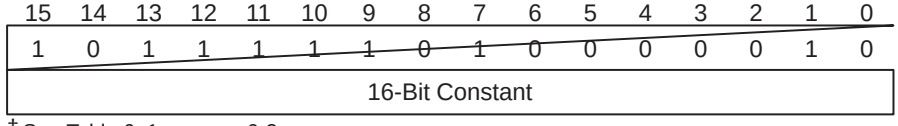


Long immediate addressing with shift



† See Table 6–1 on page 6-2.

Long immediate addressing with shift of 16



† See Table 6–1 on page 6-2.

Execution

Direct or indirect addressing:
(PC) + 1 $\rightarrow$ PC
(ACC(15–0)) **OR** (dma) $\rightarrow$ ACC(15–0)
(ACC(31–16)) $\rightarrow$ ACC(31–16)

Long immediate addressing:
(PC) + 2 $\rightarrow$ PC
(ACC) **OR** (lk $\times$ 2^{shift}) $\rightarrow$ ACC

Status Bits

Does not affect: C
Not affected by: SXM

Long immediate addressing

Description

If a long immediate constant is specified, the constant is shifted, as defined by the shift code, and zero-extended on both ends and is ORed with the contents of the accumulator (ACC). The result is stored in the ACC. If a constant is not specified, the contents of the data memory address (dma) are ORed with the contents of the accumulator low byte (ACCL). The result is stored in the ACCL and the contents of the accumulator high byte (ACCH) are unaffected.

OR is an accumulator memory reference instruction (see Table 6–4).

Words

1 (Direct or indirect addressing)

2 (Long immediate addressing)

Cycles

For the long immediate addressing modes, the OR instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

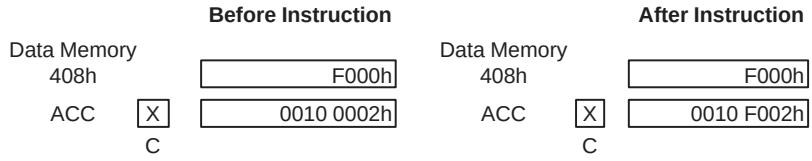
Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

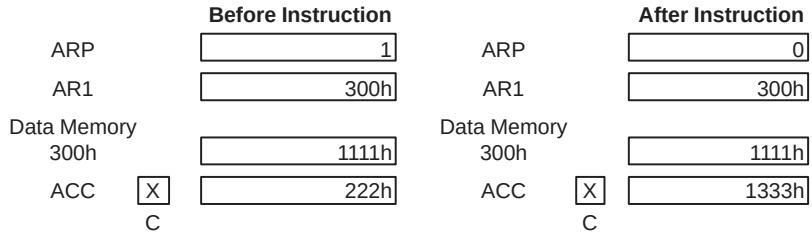
Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

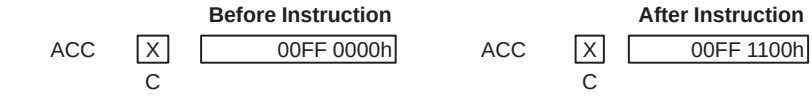
Example 1 `OR DAT8 ;(DP = 8)`



Example 2 `OR *,AR0`



Example 3 `OR #08111h,8`



Syntax **ORB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1

Execution

(PC) + 1 → PC
(ACC) **OR** (ACCB) → ACC

Status Bits None affected.

Description The contents of the accumulator (ACC) are ORed with the contents of the accumulator buffer (ACCB). The result is stored in the ACC and the contents of the ACCB are unaffected.

ORB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

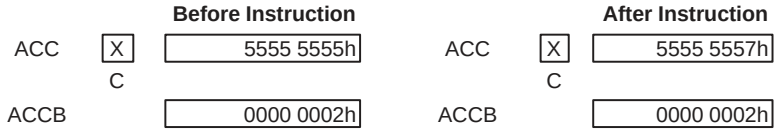
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 0RB



Syntax

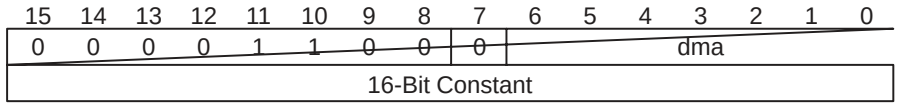
Direct: **OUT** *dma* , *PA*
Indirect: **OUT** {*ind*}, *PA* [,**AR***n*]

Operands

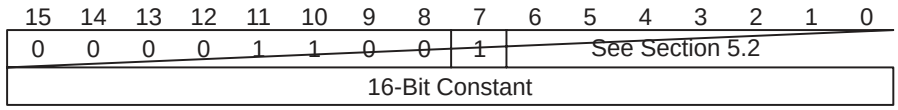
$0 \le$ dma ≤ 127
$0 \le n \le 7$
$0 \le$ port address PA ≤ 65535
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indirect addressing



Execution

(PC) + 2 $\rightarrow$ PC
While (repeat counter) $\neq 0$
Port address $\rightarrow$ address bus A15–A0
(dma) $\rightarrow$ data bus D15–D0
Port address + 1 $\rightarrow$ Port address
(repeat counter – 1) $\rightarrow$ (repeat counter)
(dma) $\rightarrow$ Port address

Status Bits

None affected.

Description

A 16-bit value from the data memory address (dma) is written to the specified I/O port. The $\overline{\text{IS}}$ line goes low to indicate an I/O access, and the $\overline{\text{STRB}}$, R/$\overline{\text{W}}$, and READY timings are the same as for an external data memory write. While port addresses 50h–5Fh are memory-mapped (see subsection 9.1.1, *Memory-Mapped Peripheral Registers and I/O Ports*); the other port addresses are not.

You can use the RPT instruction with the OUT instruction to write consecutive words in data memory to I/O space. The number of words to be moved is one greater than the number contained in the repeat counter register (RPTC) at the beginning of the instruction. When used with the RPT instruction, the OUT instruction becomes a single-cycle instruction, once the RPT pipeline is started, and the port address is incremented after each access.

OUT is an I/O and data memory operation instruction (see Table 6–9).

Words

2

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$	$5+io_{dst}+2p_{code}$
Source: SARAM	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$, $4+io_{dst}$†	$5+io_{dst}+2p_{code}$
Source: External	$3+d_{src}+io_{dst}$	$3+d_{src}+io_{dst}$	$3+d_{src}+io_{dst}$	$6+d_{src}+io_{dst}+2p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+3+nio_{dst}+2p_{code}$
Source: SARAM	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$, $3n+1+nio_{dst}$†	$3n+3+nio_{dst}+2p_{code}$
Source: External	$5n–2+nd_{src}$ $+nio_{dst}$	$5n–2+nd_{src}+$ $+nio_{dst}$	$5n–2+nd_{src}$ $+nio_{dst}$	$5n+1+nd_{src}+nio_{dst}$ $+2p_{code}$

† If the source operand and the code are in the same SARAM block

Example 1

```
OUT DAT0,57h ;(DP = 4) Output data word stored in data memory
             ;location 200h to peripheral on I/O port 57h.
```

Example 2

```
OUT *,PA15 ;Output data word referenced by current auxiliary
           ;register to peripheral on port address 15
           ;(i.e., I/O port 5Fh).
```

Syntax **PAC**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1

Execution

(PC) + 1 → PC
(shifted PREG) → ACC

Status Bits *Affected by:* PM

Description The contents of the product register (PREG) are shifted, as defined by the PM bits, and loaded into the accumulator (ACC).

PAC is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

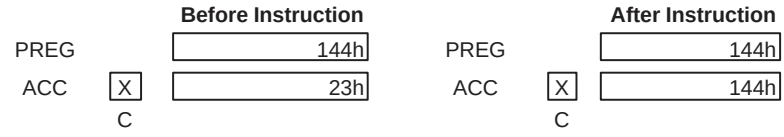
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
PAC ;(PM = 0)
```



Syntax **POP**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	0	0	1	0

Execution

(PC) + 1 → PC
(TOS) → ACC(15–0)
0 → ACC(31–16)
Pop stack one level

Status Bits None affected.

Description The contents of the top of the stack (TOS) are copied to the accumulator low byte (ACCL). The stack is popped one level after the contents are copied. The accumulator high byte (ACCH) is zero-filled.

The hardware stack is last-in, first-out with eight locations. Any time a pop occurs, every stack value is copied to the next higher stack location, and the top value is removed from the stack. After a pop, the bottom two stack words have the same value. Because each stack value is copied, if more than seven stack pops (POP, POPD, RET, RETC, RETE, or RETI instructions) occur before any pushes occur, all levels of the stack contain the same value. No provision exists to check stack underflow.

POP is a control instruction (see Table 6–10).

Words 1

Cycles

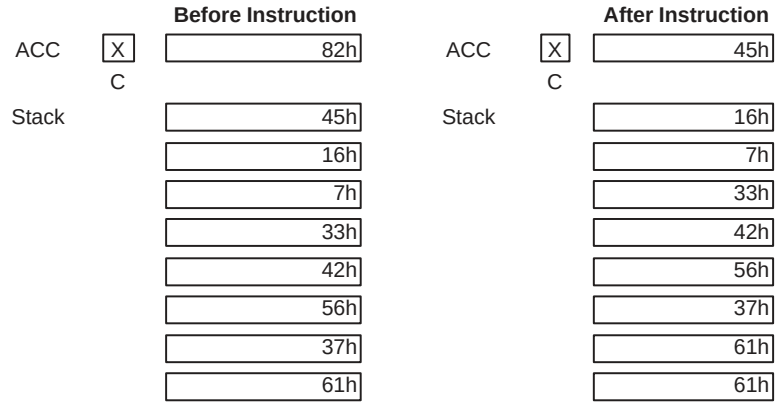
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example POP


Before Instruction
After Instruction
ACC
X
C
82h
45h
Stack
45h
16h
7h
33h
42h
56h
37h
61h
16h
7h
33h
42h
56h
37h
61h

Syntax

Direct: **POPD** *dma*
Indirect: **POPD** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(TOS) → dma
Pop stack one level

Status Bits

None affected.

Description

The contents of the top of the stack (TOS) are copied to the data memory address (dma). The values are popped one level in the lower seven locations of the stack. The value in the lowest stack location is unaffected. See the POP instruction, page 6-194, for more information.

POPD is a control instruction (see Table 6–10).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1 `POPD DAT10 ;(DP = 8)`

	Before Instruction		After Instruction
Data Memory 40Ah	55h	Data Memory 40Ah	92h
Stack	92h	Stack	72h
	72h		8h
	8h		44h
	44h		81h
	81h		75h
	75h		32h
	32h		AAh
	AAh		AAh

Example 2 `POPD *+,AR1`

	Before Instruction		After Instruction
ARP	0	ARP	1
AR0	300h	AR0	301h
Data Memory 300h	55h	Data Memory 300h	92h
Stack	92h	Stack	72h
	72h		8h
	8h		44h
	44h		81h
	81h		75h
	75h		32h
	32h		AAh
	AAh		AAh

Syntax

Direct: **PSHD** *dma*
Indirect: **PSHD** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	1	See Section 5.2						

Execution

(dma) $\rightarrow$ TOS
(PC) + 1 $\rightarrow$ PC
Push all stack locations down one level

Status Bits

None affected.

Description

The contents of the data memory address (dma) are copied to the top of the stack (TOS). The values are pushed down one level in the lower seven locations of the stack. The value in the lowest stack location is lost. See the PUSH instruction, page 6-200, for more information.

PSHD is a control instruction (see Table 6–10).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
PSHD DAT127 ;(DP = 3)
```

	Before Instruction		After Instruction
Data Memory 1FFh	65h	Data Memory 1FFh	65h
Stack	2h	Stack	65h
	33h		2h
	78h		33h
	99h		78h
	42h		99h
	50h		42h
	0h		50h
	0h		0h

Example 2

```
PSHD *,AR1
```

	Before Instruction		After Instruction
ARP	0	ARP	1
AR0	1FFh	AR0	1FFh
Data Memory 1FFh	12h	Data Memory 1FFh	12h
Stack	2h	Stack	12h
	33h		2h
	78h		33h
	99h		78h
	42h		99h
	50h		42h
	0h		50h
	0h		0h

Syntax **PUSH**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0

Execution

(PC) + 1 → PC
Push all stack locations down one level
ACC(15–0) → TOS

Status Bits None affected.

Description

The values are pushed down one level in the lower seven locations of the stack. The contents of the accumulator low byte (ACCL) are copied to the top of the stack (TOS). The values on the stack are pushed down before the ACC value is copied.

The hardware stack is last-in, first-out with eight locations. If more than eight pushes (CALA, CALL, CC, INTR, NMI, PSHD, PUSH, or TRAP instructions) occur before a pop, the first data values written are lost with each succeeding push.

PUSH is a control instruction (see Table 6–10).

Words 1

Cycles

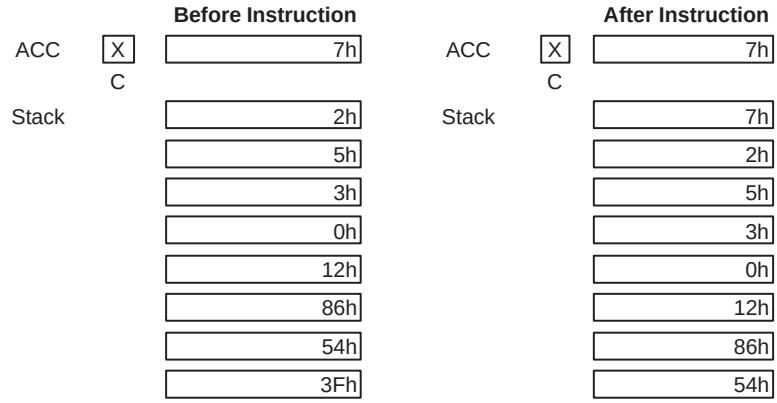
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example PUSH



Syntax **RET**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0

Execution (TOS) $\rightarrow$ PC
Pop stack one level

Status Bits None affected.

Description The contents of the top of the stack (TOS) are copied to the program counter (PC). The stack is popped one level after the contents are copied. The RET instruction is used with the CALA, CALL, and CC instructions for subroutines.

RET is a branch and call instruction (see Table 6–8).

Words 1

Cycles The RET instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example RET

	Before Instruction		After Instruction
PC	96h	PC	37h
Stack	37h	Stack	45h
	45h		75h
	75h		21h
	21h		3Fh
	3Fh		45h
	45h		6Eh
	6Eh		6Eh
	6Eh		6Eh

Syntax

RETC *cond* [, *cond1*] [,...]

Operands

Conditions:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
$\overline{\text{BIO}}$ low	BIO
TC = 0	NTC
TC = 1	TC
Unconditional	UNC

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	TP †		ZLVC †				ZLVC †			

† See Table 6–1 on page 6-2.

Execution

If (condition(s)):
(TOS) → PC
Pop stack one level
Else, continue

Status Bits

None affected.

Description

If the specified conditions are met, the contents of the top of the stack (TOS) are copied to the program counter (PC). The stack is popped one level after the contents are copied. Not all combinations of the conditions are meaningful. If the specified conditions are not met, control is passed to the next instruction.

RETC is a branch and call instruction (see Table 6–8).

Words

1

Cycles

The RETC instruction is not repeatable.

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+3p†
False	2	2	2	2+p

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
RETC GEQ,NOV ;A return, RET, is executed if the
             ;accummulator contents are positive and the
             ;OV bit is a zero.
```

Syntax **RETCD** *cond* [, *cond1*] [,...]

Operands

Conditions:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
$\overline{\text{BIO}}$ low	BIO
TC = 0	NTC
TC = 1	TC
Unconditional	UNC

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	TP †		ZLVC †				ZLVC †			

† See Table 6–1 on page 6-2.

Execution

If (condition(s)):
(TOS) → PC
Pop stack one level
Else, continue

Status Bits None affected.

Description

The one 2-word instruction or two 1-word instructions following the RETCD instruction are fetched from program memory and executed before the execution of the return. The two instruction words following the RETCD instruction have no effect on the conditions being tested.

After the instructions are executed if the specified conditions are met, the contents of the top of the stack (TOS) are copied to the program counter (PC). The stack is popped one level after the contents are copied. Not all combinations of the conditions are meaningful. If the specified conditions are not met, control is passed to the next instruction.

RETCD is a branch and call instruction (see Table 6–8).

Words 1

Cycles The RETCD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

Example

```
RETCD C      ;A return, RET, is executed if the carry
MAR *,4      ;bit is set. The two instructions following
LAR AR3,#1h ;the return instruction are executed
             ;before the return is taken.
```

Syntax **RETD**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Execution (TOS) $\rightarrow$ PC
Pop stack one level

Status Bits None affected.

Description The one 2-word instruction or two 1-word instructions following the RETD instruction are fetched from program memory and executed before the execution of the return.

After the instructions are executed the contents of the top of the stack (TOS) are copied to the program counter (PC). The stack is popped one level after the contents are copied.

RETD is a branch and call instruction (see Table 6–8).

Words 1

Cycles The RETD instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

Example

```
RETD
MAR *,4
LACC #1h
```

	Before Instruction		After Instruction
PC	96h	PC	37h
ARP	0	ARP	4
ACC	0h	ACC	01h
Stack	37h	Stack	45h
	45h		75h
	75h		21h
	21h		3Fh
	3Fh		45h
	45h		6Eh
	6Eh		6Eh
	6Eh		6Eh

Syntax **RETE**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	0	1	0

Execution

(TOS) $\rightarrow$ PC
Pop stack one level
0 $\rightarrow$ INTM

Status Bits *Affects:* ARB, ARP, AVIS, BRAF, C, CNF, DP, HM, INTM, MP/$\overline{\text{MC}}$, NDX, OV, OVLY, OVM, PM, RAM, SXM, TC, TRM, and XF

Description The contents of the top of the stack (TOS) are copied to the program counter (PC). The stack is popped one level after the contents are copied. The RETE instruction automatically clears the INTM bit and pops the shadow register values (see the RETI description, page 6-209).

The RETE instruction is the equivalent of clearing the INTM bit and executing a RETI instruction.

RETE is a branch and call instruction (see Table 6–8).

Words 1

Cycles The RETE instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example RETE

	Before Instruction		After Instruction
PC	96h	PC	37h
ST0	xx6xh	ST0	xx4xh
Stack	37h	Stack	45h
	45h		75h
	75h		21h
	21h		3Fh
	3Fh		45h
	45h		6Eh
	6Eh		6Eh
	6Eh		6Eh

Syntax **RETI**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0

Execution (TOS) → PC
Pop stack one level

Status Bits

Affects:
ARB, ARP, AVIS, BRAF, C, CNF, DP, HM, MP/$\overline{\text{MC}}$, NDX, OV, OVLY, OVM, PM, RAM, SXM, TC, TRM, and XF

Does not affect:
INTM

Description The contents of the top of the stack (TOS) are copied to the program counter (PC). The values in the shadow registers (stored when an interrupt was taken) are returned to their corresponding strategic registers. The following registers are shadowed: ACC, ACCB, ARCR, INDX, PMST, PREG, ST0, ST1, TREG0, TREG1, and TREG2. The INTM bit in ST0 and the XF bit in ST1 are not saved or restored to or from the shadow registers during an interrupt service routine (ISR).

RETI is a branch and call instruction (see Table 6–8).

Words 1

Cycles The RETI instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
RETI
```

	Before Instruction		After Instruction
PC	96h	PC	37h
Stack	37h	Stack	45h
	45h		75h
	75h		21h
	21h		3Fh
	3Fh		45h
	45h		6Eh
	6Eh		6Eh
	6Eh		6Eh

Syntax **ROL**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0

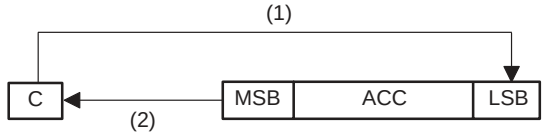
Execution

(PC) + 1 → PC
C → ACC(0)
(ACC(31)) → C
(ACC(30–0)) → ACC(31–1)

Status Bits

Not affected by:	*Affects*:
SXM	C

Description The contents of the accumulator (ACC) are rotated left 1 bit. The value of the C bit is shifted into the LSB of the ACC. The MSB of the original ACC is shifted into the C bit.



ROL is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

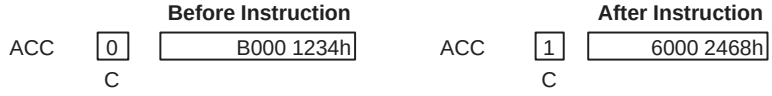
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
ROL
```



Syntax **ROLB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	0	0

Execution

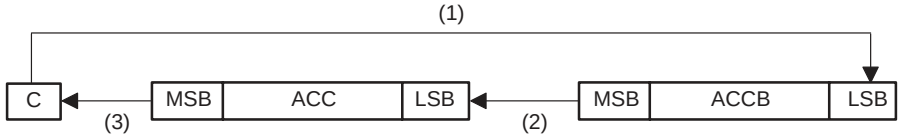
(PC) + 1 → PC
C → ACCB(0)
(ACCB(30–0)) → ACCB(31–1)
(ACCB(31)) → ACC(0)
(ACC(30–0)) → ACC(31–1)
(ACC(31)) → C

Status Bits

Not affected by:	*Affects*:
SXM	C

Description

The ROLB instruction causes a 65-bit rotation. The contents of both the accumulator (ACC) and accumulator buffer (ACCB) are rotated left 1 bit. The value of the C bit is shifted into the LSB of the ACCB. The MSB of the original ACCB is shifted into the LSB of the ACC. The MSB of the original ACC is shifted into the C bit.



ROLB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
ROLB
```

		Before Instruction			After Instruction
ACC	1	0808 0808h	ACC	0	1010 1011h
	C			C	
ACCB		FFFF FFFEh	ACCB		FFFF FFFDh

Syntax **ROR**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	1

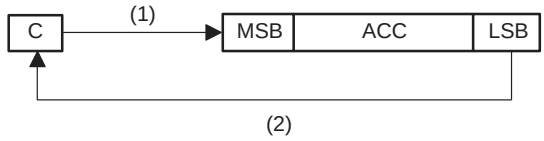
Execution

(PC) + 1 → PC
C → ACC(31)
(ACC(0)) → C
(ACC(31–1)) → ACC(30–0)

Status Bits

Not affected by:	*Affects*:
SXM	C

Description The contents of the accumulator (ACC) are rotated right 1 bit. The value of the C bit is shifted into the MSB of the ACC. The LSB of the original ACC is shifted into the C bit.



ROR is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

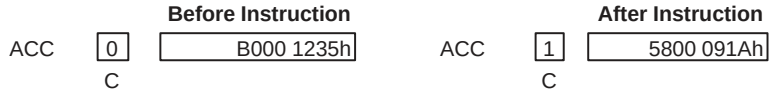
Cycle Timings for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example R0R



Syntax **RORB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	0	1

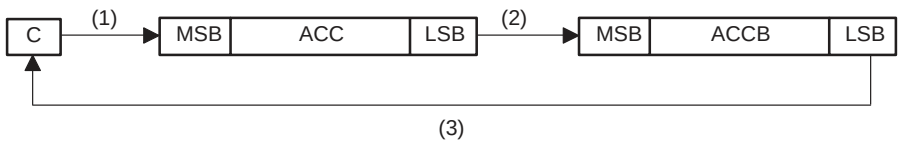
Execution

(PC) + 1 → PC
C → ACC(31)
(ACC(31–1)) → ACC(30–0)
(ACC(0)) → ACCB(31)
(ACCB(31–1)) → ACCB(30–0)
(ACCB(0)) → C

Status Bits

Not affected by:	*Affects*:
SXM	C

Description The RORB instruction causes a 65-bit rotation. The contents of both the accumulator (ACC) and accumulator buffer (ACCB) are rotated right 1 bit. The value of the C bit is shifted into the MSB of the ACC. The LSB of the original ACC is shifted into the MSB of the ACCB. The LSB of the original ACCB is shifted into the C bit.



RORB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

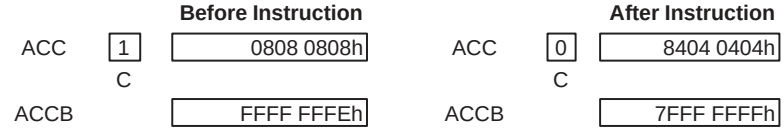
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example R0RB



Syntax

Direct: **RPT** *dma*
Indirect: **RPT** {*ind*} [,**AR***n*]
Short immediate: **RPT** #*k*
Long immediate: **RPT** #*lk*

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le k \le 255$
$0 \le lk \le 65535$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	8-Bit Constant							

Long immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	0	0	0	1	0	0
16-Bit Constant															

Execution

Direct or indirect addressing:
(PC) + 1 → PC
(dma) → RPTC

Short immediate addressing:
(PC) + 1 → PC
k → RPTC

Long immediate addressing:
(PC) + 2 → PC
lk → RPTC

Status Bits

None affected.

Description

The contents of the data memory address (dma), an 8-bit constant, or a 16-bit constant are loaded into the repeat counter register (RPTC). The instruction following the RPT instruction is repeated *n* times, where *n* is one more than the initial value of the RPTC.

Since the RPTC cannot be saved during a context switch, repeat loops are regarded as multicycle instructions and are not interruptible. However, the processor can halt a repeat loop in response to an external $\overline{\text{HOLD}}$ signal. The execution restarts when $\overline{\text{HOLD}}$/$\overline{\text{HOLDA}}$ are deasserted. The RPTC is cleared on a device reset.

The RPT instruction is especially useful for block moves, multiply-accumulates, normalization, and other functions. RPT is a control instruction (see Table 6–10).

Words

1 (Direct, indirect, or short immediate addressing)

2 (Long immediate addressing)

Cycles

The RPT instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

Example 1

```
RPT DAT127 ;(DP = 31)
```

	Before Instruction		After Instruction
Data Memory 0FFFh	0Ch	Data Memory 0FFFh	0Ch
RPTC	0h	RPTC	0Ch

Example 2

```
RPT *,AR1
```

	Before Instruction		After Instruction
ARP	0	ARP	1
AR0	300h	AR0	300h
Data Memory 300h	0FFFh	Data Memory 300h	0FFFh
RPTC	0h	RPTC	0FFFh

Example 3

```
RPT #1 ;Repeat next instruction 2 times.
```

	Before Instruction		After Instruction
RPTC	0h	RPTC	1h

Example 4

```
RPT #1111h ;Repeat next instruction 4370 times.
```

	Before Instruction		After Instruction
RPTC	0h	RPTC	1111h

Syntax **RPTB** *pma*

Operands $0 \leq pma \leq 65535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	0	0	0	1	1	0
16-Bit Constant															

Execution

1 → BRAF
(PC) + 2 → PASR
pma → PAER

Status Bits

Affected by:	*Affects*:
BRAF	BRAF

Description

A block of instructions to be repeated a number of times is specified by the memory-mapped block repeat counter register (BRCR) without any penalty for looping. The BRCR must be loaded before execution of an RPTB instruction. When the RPTB instruction is executed, the BRAF bit is set, the block repeat program address start register (PASR) is loaded with the contents of the program counter (PC) + 2, and the block repeat program address end register (PAER) is loaded with the program memory address (pma). Block repeat can be deactivated by clearing the BRAF bit. The number of loop iterations is given as (BRCR) + 1.

The RPTB instruction is interruptible. However, RPTB instructions cannot be nested unless the BRAF bit is properly set and the BRCR, PAER, and PASR are appropriately saved and restored. Single-instruction repeat loops (RPT and RPTZ) can be included as part of RPTB blocks.

Note:

The repeat block must contain at least 3 instruction words for proper operation.

RPTB is a control instruction (see Table 6–10).

Words 2

Cycles

The RPTB instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

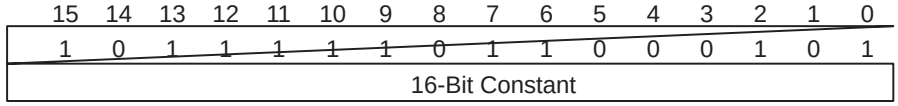
Example

```
  SPLK  #iterations_minus_1,BRCR   ;initialize BRCR
  RPTB  end_block – 1
  LACC  DAT1
  ADD   DAT2
  SACL  DAT1
end_block
```

Syntax **RPTZ #*lk***

Operands 0 ≤ lk ≤ 65535

Opcode



Execution

0 → ACC
0 → PREG
(PC) + 1 → PC
lk → RPTC

Status Bits None affected.

Description The contents of the accumulator (ACC) and product register (PREG) are cleared. The 16-bit constant, *lk*, is loaded into the repeat counter register (RPTC). The instruction following the RPTZ instruction is repeated *lk* + 1 times. The RPTZ instruction is equivalent to the following instruction sequence:

```
MPY #0
PAC
RPT #<lk>
```

RPTZ is a control instruction (see Table 6–10).

Words 2

Cycles The RPTZ instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

Example

```
RPTZ #7FFh  ;Zero product register and accumulator.
MACD pma,*+  ;Repeat MACD 2048 times.
```

Syntax **SACB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	0

Execution

(PC) + 1 → PC
(ACC) → ACCB

Status Bits None affected.

Description The contents of the accumulator (ACC) are copied to the accumulator buffer (ACCB). The contents of the ACC are unaffected.

SACB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example SACB

	Before Instruction		After Instruction
ACC	7C63 8421h	ACC	7C63 8421h
ACCB	5h	ACCB	7C63 8421h

Syntax

Direct: **SACH** *dma* [,*shift2*]
Indirect: **SACH** {*ind*} [,*shift2*[,**AR***n*]]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le shift2 \le 7$ (defaults to 0)
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	SHF †			0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	SHF †			1	See Section 5.2						

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) $\times$ 2^{shift2} $\rightarrow$ dma

Status Bits

Not affected by: SXM

Description

The contents of the accumulator (ACC) are shifted left from 0 to 7 bits, as defined by the shift code, and the high-order bits are stored in the data memory address (dma). During shifting, the high-order bits are lost. The contents of the ACC are unaffected.

SACH is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
SACH DAT10,1 ;(DP = 4)
```

	Before Instruction	After Instruction
ACC	0420 8001h	0420 8001h
C	X	X
Data Memory 20Ah	0h	0841h

Example 2

```
SACH *+,0,AR2
```

	Before Instruction	After Instruction
ARP	1	2
AR1	300h	301h
ACC	0420 8001h	0420 8001h
C	X	X
Data Memory 300h	0h	0420h

Syntax

Direct: **SACL** *dma* [,*shift2*]
Indirect: **SACL** {ind} [,*shift2*[,**AR***n*]]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
$0 \le shift2 \le 7$ (defaults to 0)
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	SHF †			0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	SHF †			1	See Section 5.2						

† See Table 6–1 on page 6-2.

Execution

$(PC) + 1 \rightarrow PC$
$(ACC(15–0)) \times 2^{shift2} \rightarrow dma$

Status Bits

Not affected by: SXM

Description

The contents of the accumulator low byte (ACCL) are shifted left from 0 to 7 bits, as defined by the shift code, and stored in the data memory address (dma). During shifting, the low-order bits are zero-filled and the high-order bits are lost. The contents of the ACC are unaffected.

SACL is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1 `SACL DAT11,1 ;(DP = 4)`

	Before Instruction	After Instruction
ACC	7C63 8421h	7C63 8421h
X		
C		
Data Memory 20Bh	05h	0842h

Example 2 `SACL *,0,AR7`

	Before Instruction	After Instruction
ARP	6	7
AR6	300h	300h
ACC	00FF 8421h	00FF 8421h
X		
C		
Data Memory 300h	05h	8421h

Syntax

Direct: **SAMM** *dma*
Indirect: **SAMM** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(ACC(15–0)) → dma(0–7)

Status Bits

None affected.

Description

The contents of the accumulator low byte (ACCL) are copied to the addressed memory-mapped register. The 9 MSBs of the data memory address are cleared, regardless of the current value of the data memory page pointer (DP) bits or the upper 9 bits of the current AR. The SAMM instruction allows the ACC to be stored to any memory location on data memory page 0 without modifying the DP bits.

SAMM is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
MMR†	1	1	1	1+p
MMPORT	$2+io_{dst}$	$2+io_{dst}$	$2+io_{dst}$	$4+io_{dst}$

† Add one more cycle if source is a peripheral memory-mapped register access

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
MMR‡	n	n	n	n+p
MMPORT	$2+nio_{dst}$	$2+nio_{dst}$	$2+nio_{dst}$	$2n+2+p+p\ nio_{dst}$

‡ Add *n* more cycles if source is a peripheral memory-mapped register access

Example 1 `SAMM PRD ;(DP = 6)`

	Before Instruction		After Instruction
ACC	80h	ACC	80h
PRD	05h	PRD	80h
Data Memory 325h	0Fh	Data Memory 325h	0Fh

Example 2 `SAMM *,AR2 ;(BMAR = 1Fh)`

	Before Instruction		After Instruction
ARP	7	ARP	2
AR7	31Fh	AR7	31Fh
ACC	080h	ACC	080h
BMAR	0h	BMAR	080h
Data Memory 31Fh	11h	Data Memory 31Fh	11h

Syntax

Direct: **SAR AR***x*, *dma*
Indirect: **SAR AR***x*,{ind} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le x \le 7$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	ARX †			0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	ARX †			1	See Section 5.2						

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 → PC
(AR) → dma

Status Bits

Affected by: NDX

Description

The contents of the auxiliary register (AR) are stored in the data memory address (dma). When the contents of the current AR are modified in the indirect addressing mode, the SAR instruction stores the value of the AR contents before it is incremented, decremented, or indexed by the contents of the index register (INDX).

You can maintain software compatibility with the ’C2x by clearing the NDX bit. This causes any ’C2x instruction that loads AR0 to load the auxiliary register compare register (ARCR) and INDX also, maintaining ’C5x object-code compatibility with the ’C2x.

The SAR and LAR (load auxiliary register) instructions can be used to store and load the ARs during subroutine calls and interrupts. If an AR is not being used for indirect addressing, LAR and SAR enable the register to be used as an additional storage register, especially for swapping values between data memory locations without affecting the contents of the accumulator (ACC).

SAR is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
SAR AR0,DAT30 ;(DP = 6)
```

	Before Instruction		After Instruction
AR0	37h	AR0	37h
Data Memory 31Eh	18h	Data Memory 31Eh	37h

Example 2

```
SAR AR0,*+
```

	Before Instruction		After Instruction
AR0	401h	AR0	402h
Data Memory 401h	0h	Data Memory 401h	401h

Syntax **SATH**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	1	0

Execution

(PC) + 1 $\rightarrow$ PC
16 $\times$ (TREG1(4)) $\rightarrow$ count
(ACC) right-shifted by count $\rightarrow$ ACC

Status Bits

Affected by:	*Does not affect*:
SXM	C

Description

The SATH instruction, in conjunction with the SATL instruction, allows for a 2-cycle 0- to 31-bit shift right. The contents of the accumulator (ACC) are barrel-shifted right 16 bits as defined by bit 4 of TREG1. If bit 4 of TREG1 is set, the contents of the ACC are barrel-shifted right by 16 bits. If bit 4 of TREG1 is cleared, the contents of the ACC are unaffected.

If the SXM bit is cleared, the high-order bits are zero-filled; if the SXM bit is set, the high-order bits of the ACC are filled with copies of ACC bit 31. The C bit is unaffected.

SATH is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

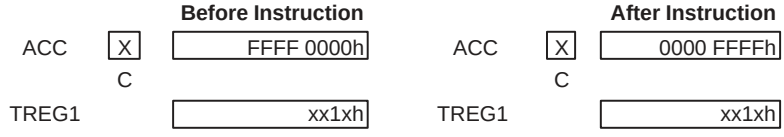
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1

```
SATH ;(SXM = 0)
```



Example 2

```
SATH ;(SXM = 1)
```

		Before Instruction			After Instruction
ACC	X	FFFF 0000h		ACC	X	FFFF FFFFh
	C				C	
TREG1		xx1xh		TREG1		xx1xh

Syntax **SATL**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	1	1

Execution

(PC) + 1 → PC
(TREG1(3–0)) → count
(ACC) right-shifted by count → ACC

Status Bits

Affected by:	*Does not affect*:
SXM	C

Description

The SATL instruction, in conjunction with the SATH instruction, allows for a 2-cycle 0- to 31-bit shift right. The contents of the accumulator (ACC) are barrel-shifted right 0 to 15 bits as defined by the 4 LSBs of TREG1.

If the SXM bit is cleared, the high-order bits are zero-filled; if the SXM bit is set, the high-order bits of the ACC are filled with copies of ACC bit 31. The C bit is unaffected.

SATL is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

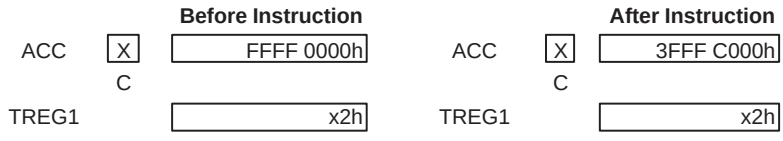
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

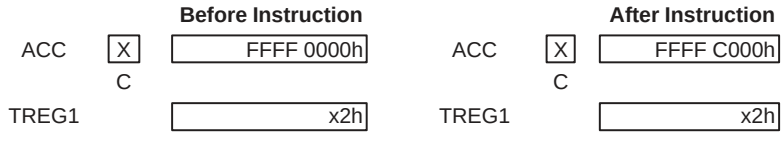
Example 1

```
SATL ;(SXM = 0)
```



Example 2

```
SATL ;(SXM = 1)
```



Syntax **SBB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	0	0

Execution

(PC) + 1 → PC
(ACC) – (ACCB) → ACC

Status Bits *Affects:* C

Description The contents of the accumulator buffer (ACCB) are subtracted from the contents of the accumulator (ACC). The result is stored in the ACC and the contents of the ACCB are unaffected. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

SBB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

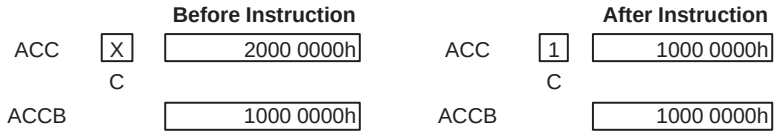
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example SBB



Syntax **SBBB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	0	1

Execution

(PC) + 1 → PC
(ACC) – (ACCB) – (logical inversion of C) → ACC

Status Bits *Affects:* C

Description The contents of the accumulator buffer (ACCB) and the logical inversion of the C bit are subtracted from the contents of the accumulator (ACC). The result is stored in the ACC and the contents of the ACCB are unaffected. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

SBBB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

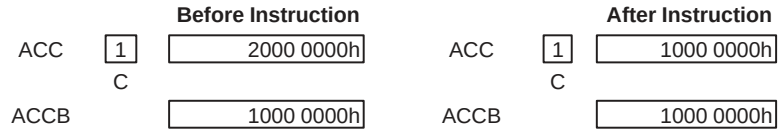
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

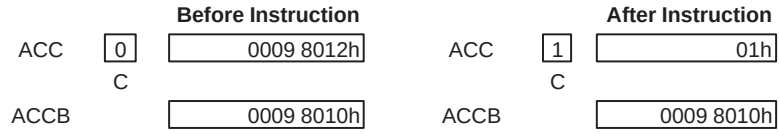
Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example 1 SBBB



Example 2 SBBB



Syntax **SBRK** #*k*

Operands $0 \leq k \leq 255$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	8-Bit Constant							

Execution (PC) + 1 → PC
(current AR) – 8-bit positive constant → current AR

Status Bits None affected.

Description The 8-bit immediate value, right-justified, is subtracted from the current auxiliary register (AR). The result is stored in the current AR. The subtraction takes place in the auxiliary register arithmetic unit (ARAU), with the immediate value treated as a 8-bit positive integer.

SBRK is an auxiliary registers and data memory page pointer instruction (see Table 6–5).

Words 1

Cycles The SBRK instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Example

```
SBRK #0FFh
```

	Before Instruction		After Instruction
ARP	7	ARP	7
AR7	0h	AR7	FF01h

Syntax **SETC** *control bit*

Operands *control bit* : {C, CNF, HM, INTM, OVM, SXM, TC, XF}

Opcode

SETC OVM (Set overflow mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	1	1

SETC SXM (Set sign extension mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	1	1

SETC HM (Set hold mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	0	1

SETC TC (Set test/control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	1	1

SETC C (Set carry)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	1

SETC XF (Set external flag pin)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	0	1

SETC CNF (Set configuration control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	0	1

SETC INTM (Set interrupt mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	1

Execution

(PC) + 1 $\rightarrow$ PC
1 $\rightarrow$ control bit

Status Bits Affects selected control bit.

Description The specified control bit is set. The LST instruction can also be used to load ST0 and ST1. See Section 4.4, *Status and Control Registers*, for more information on each control bit.

SETC is a control instruction (see Table 6–10).

An IDLE instruction must not follow a SETC INTM instruction; otherwise, an unmasked interrupt may take the device out of idle before the INTM bit is set.

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
SETC TC ;TC is bit 11 of ST1
```

	Before Instruction		After Instruction
ST1	x1xxh	ST1	x9xxh

Syntax **SFL**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	0	0	1

Execution

(PC) + 1 → PC
(ACC(31)) → C
(ACC(30–0)) → ACC(31–1)
0 → ACC(0)

Status Bits

Not affected by:	*Affects*:
SXM	C

Description The contents of the accumulator (ACC) are shifted left 1 bit. The MSB of the ACC is shifted into the C bit. The LSB of the ACC is filled with a 0. The SFL instruction, unlike the SFR instruction, is unaffected by the SXM bit.

SFL is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

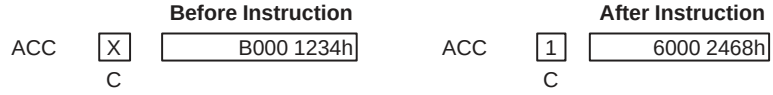
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example SFL



Syntax **SFLB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	1	0

Execution

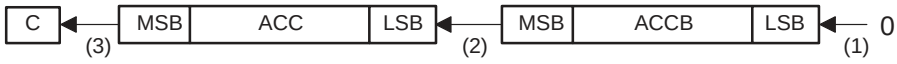
(PC) + 1 → PC
0 → ACCB(0)
(ACCB(30–0)) → ACCB(31–1)
(ACCB(31)) → ACC(0)
(ACC(30–0)) → ACC(31–1)
(ACC(31) → C

Status Bits

Not affected by:	*Affects*:
SXM	C

Description

The contents of both the accumulator (ACC) and accumulator buffer (ACCB) are shifted left 1 bit. The LSB of the ACCB is filled with a 0, and the MSB of the ACCB is shifted into the LSB of the ACC. The MSB of the ACC is shifted into the C bit. The SFLB instruction, unlike the SFRB instruction, is unaffected by the SXM bit.



SFLB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

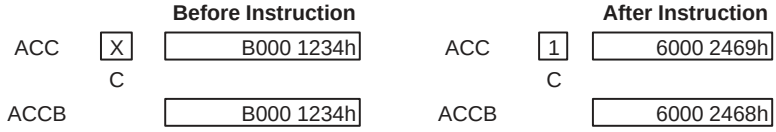
Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example SFLB



Syntax **SFR**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	0	1	0

Execution

(PC) + 1 → PC

If SXM = 0:
0 → ACC(31)

If SXM = 1
(ACC(31)) → ACC(31)

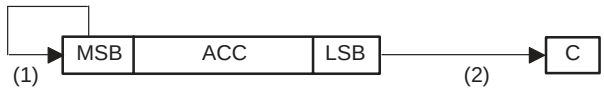
(ACC(31–1)) → ACC(30–0)
(ACC(0)) → C

Status Bits

Affected by:	*Affects*:
SXM	C

Description The contents of the accumulator (ACC) are shifted right 1 bit. The type of shift is determined by the SXM bit. If the SXM bit is cleared, the SFR instruction produces a logic right shift. The MSB of the ACC is filled with a 0. The LSB of the ACC is shifted into the C bit.

If the SXM bit is set, the SFR instruction produces an arithmetic right shift. The MSB (sign bit) of the ACC is unchanged and is copied into ACC bit 30. The LSB of the ACC is shifted into the C bit.



SFR is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

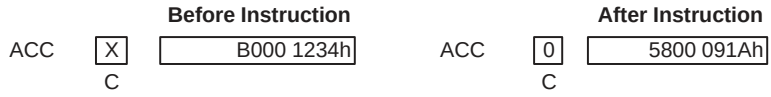
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

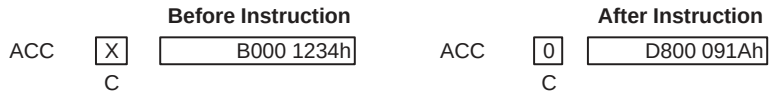
Example 1

```
SFR ;(SXM = 0)
```



Example 2

```
SFR ;(SXM = 1)
```



Syntax **SFRB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	1	1

Execution

(PC) + 1 → PC

If SXM=0:
0 → ACC(31)

If SXM=1:
(ACC(31)) → ACC(31)

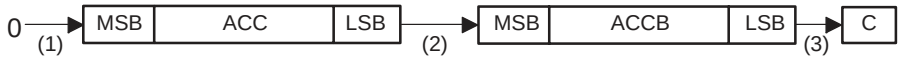
(ACC(31–1)) → ACC(30–0)
(ACC(0)) → ACCB (31)
(ACCB(31–1)) → ACCB(30–0)
(ACCB(0)) → C

Status Bits

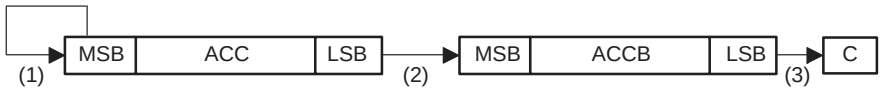
Affected by:	*Affects*:
SXM	C

Description

The contents of both the accumulator (ACC) and accumulator buffer (ACCB) are shifted right 1 bit. The type of shift is determined by the SXM bit. If the SXM bit is cleared, the SFR instruction produces a logic right shift. The MSB of the ACC is filled with a 0. The LSB of the ACC is shifted into the MSB of the ACCB. The LSB of the ACCB is shifted into the C bit.



If the SXM bit is set, the SFR instruction produces an arithmetic right shift. The MSB (sign bit) of the ACC is unchanged and is copied into ACC bit 30. The LSB of the ACC is shifted into the MSB of the ACCB. The LSB of the ACCB is shifted into the C bit.



SFRB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

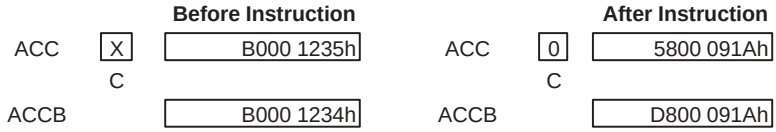
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

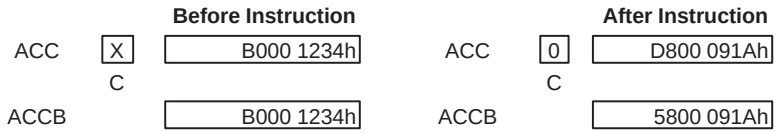
Example 1

```
SFRB ;(SXM = 0)
```



Example 2

```
SFRB ;(SXM = 1)
```



Syntax

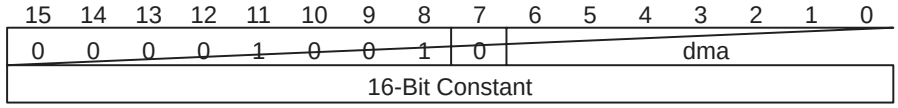
Direct: **SMMR** *dma*, #*addr*
Indirect: **SMMR** {*ind*}, #*addr* [,**AR***n*]

Operands

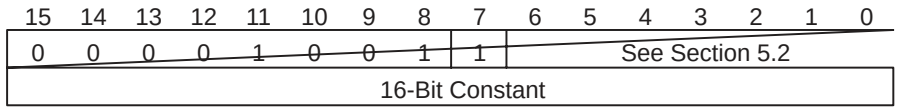
$0 \le addr \le 65535$
$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indindirect addressing



Execution

PFC $\rightarrow$ MCS
(PC) + 2 $\rightarrow$ PC
lk $\rightarrow$ PFC
While (repeat counter $\neq$ 0):
    (src, specified by lower 7 bits of dma) $\rightarrow$ (dst, addressed by PFC)
    (PFC) + 1 $\rightarrow$ PFC
    (repeat counter) – 1 $\rightarrow$ repeat counter
MCS $\rightarrow$ PFC

Status Bits

None affected.

Description

The memory-mapped register value pointed at by the lower 7 bits of the data memory address (dma) is stored to the data memory location addressed by the 16-bit source address, *#addr*. The 9 MSBs of the dma are cleared, regardless of the current value of the data memory page pointer (DP) bits or the upper 9 bits of the current AR. The SMMR instruction allows any memory location on data memory page 0 to be stored anywhere in data memory without modifying the DP bits.

When using the SMMR instruction with the RPT instruction, the destination address, *#addr*, is incremented after every memory-mapped store operation.

SMMR is an I/O and data memory operation instruction (see Table 6–9).

Words

2

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM Source: MMR‡	2	2	2	$2+2p_{code}$
Destination: SARAM Source: MMR‡	2	2	2, 3†	$2+2p_{code}$
Destination: External Source: MMR‡	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+2p_{code}$
Destination: DARAM Source: MMPORT	$3+io_{src}$	$3+io_{src}$	$3+io_{src}$	$4+io_{src}+2p_{code}$
Destination: SARAM Source: MMPORT	$3+io_{src}$	$3+io_{src}$	$3+io_{src}$, $4+io_{src}$†	$3+io_{src}+2p_{code}$
Destination: External Source: MMPORT	$4+io_{src}+d_{dst}$	$4+io_{src}+d_{dst}$	$4+io_{src}+d_{dst}$	$6+io_{src}+d_{dst}+2p_{code}$

† If the destination operand and the code are in the same SARAM block
‡ Add one more cycle for peripheral memory-mapped register access.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM Source: MMR§	2n	2n	2n	$2n+2p_{code}$
Destination: SARAM Source: MMR§	2n	2n	2n, 2n+2†	$2n+2p_{code}$
Destination: External Source: MMR§	$3n+nd_{dst}$	$3n+nd_{dst}$	$3n+nd_{dst}$	$3n+3+nd_{dst}+2p_{code}$
Destination: DARAM Source: MMPORT	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$	$2n+1+nio_{src}+2p_{code}$
Destination: SARAM Source: MMPORT	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$, $2n+2+nio_{src}$†	$2n+1+nio_{src}+2p_{code}$
Destination: External Source: MMPORT	$5n-2+nd_{dst}$ $+nio_{src}$	$5n-2+nd_{dst}$ $+nio_{src}$	$5n-2+nd_{dst}$ $+nio_{src}$	$5n+1+nd_{dst}+nio_{src}$ $+2p_{code}$

† If the destination operand and the code are in the same SARAM block
§ Add *n* more cycles for peripheral memory-mapped register access.

Example 1

```
SMMR CBCR,#307h ;(DP = 6, CBCR = 1Eh)
```

	Before Instruction		After Instruction
Data Memory 307h	1376h	Data Memory 307h	5555h
CBCR	5555h	CBCR	5555h

Example 2

```
SMMR *,#307h,AR6 ;(CBCR = 1Eh)
```

	Before Instruction		After Instruction
ARP	6	ARP	6
AR6	F01Eh	AR6	F01Eh
Data Memory 307h	1376h	Data Memory 307h	5555h
CBCR	5555h	CBCR	5555h

Syntax **SPAC**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	1

Execution

(PC) + 1 → PC
(ACC) – (shifted PREG) → ACC

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM and PM	SXM	C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and subtracted from the contents of the accumulator (ACC). The result is stored in the ACC. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set. The SPAC instruction is not affected by the SXM bit and the PREG is always sign extended.

The SPAC instruction is a subset of the LTS, MPYS, and SQRS instructions. SPAC is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

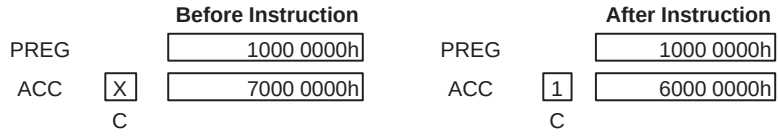
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example

```
SPAC ;(PM = 0)
```



Syntax

Direct: **SPH** *dma*
Indirect: **SPH** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(PREG shifter output (31–16)) → dma

Status Bits

Affected by: PM

Description

The contents of the product register (PREG) high byte are shifted, as defined by the PM bits, and stored in the data memory address (dma). The contents of the PREG and the accumulator (ACC) are unaffected. When the right-shift-by-6 mode (PM is set to 11_2) is selected, high-order bits are sign extended. When left shifts are selected, low-order bits are filled from the PREG low byte.

SPH is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
SPH DAT3 ;(DP = 4, PM = 0)
```

	Before Instruction		After Instruction
PREG	FE07 9844h	PREG	FE07 9844h
203h	4567h	203h	FE07h

Example 2

```
SPH *,AR7 ;(PM = 2)
```

	Before Instruction		After Instruction
ARP	6	ARP	7
AR6	203h	AR6	203h
PREG	FE07 9844h	PREG	FE07 9844h
Data Memory 203h	4567h	Data Memory 203h	E079h

Syntax

Direct: **SPL** *dma*
Indirect: **SPL** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(PREG shifter output (15–0)) → dma

Status Bits

Affected by: PM

Description

The contents of the product register (PREG) low byte are shifted, as defined by the PM bits, and stored in the data memory address (dma). The contents of the PREG and the accumulator (ACC) are unaffected. When the right-shift-by-6 mode (PM is set to 11_2) is selected, high-order bits are filled from the PREG high byte. When left shifts are selected, low-order bits are zero-filled.

SPL is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
SPL DAT5 ;(DP = 1, PM = 2)
```

	Before Instruction		After Instruction
PREG	FE07 9844h	PREG	FE07 9844h
Data Memory 205h	4567h	Data Memory 205h	8440h

Example 2

```
SPL *,AR3 ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	2	ARP	3
AR2	205h	AR2	205h
PREG	FE07 9844h	PREG	FE07 9844h
Data Memory 205h	4567h	Data Memory 205h	9844h

Syntax

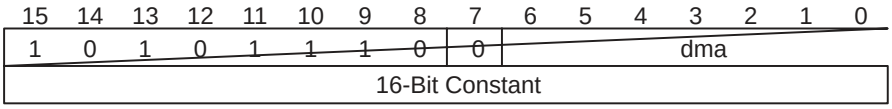
Direct: **SPLK** #*lk*, *dma*
Indirect: **SPLK** #*lk*, {*ind*} [,**AR***n*]

Operands

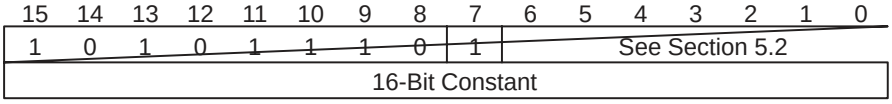
$0 \le dma \le 127$
$0 \le n \le 7$
lk: 16-bit constant
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing



Indirect addressing



Execution

(PC) + 2 → PC
lk → dma

Status Bits

None affected.

Description

The 16-bit constant is stored into the data memory address (dma). The parallel logic unit (PLU) supports this bit manipulation independently of the arithmetic logic unit (ALU), so the contents of the accumulator (ACC) are unaffected.

SPLK is a parallel logic unit (PLU) instruction (see Table 6–6).

Words

2

Cycles

The SPLK instruction is not repeatable.

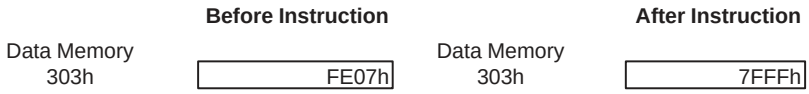
Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2, 3†	2+2p
External	3+d	3+d	3+d	5+d+2p

† If the operand and the code are in the same SARAM block

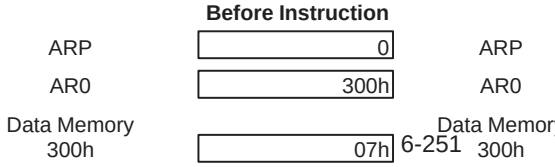
Example 1

```
SPLK #7FFFh,DAT3 ;(DP = 6)
```



Example 2

```
SPLK #1111h,*+,AR4
```



Syntax **SPM** *constant*

Operands $0 \leq \text{constant} \leq 3$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	0	0	0	0	PM †	

† See Table 6–1 on page 6-2.

Execution

(PC) + 1 → PC
Constant → PM

Status Bits

Not affected by:	*Affects*:
SXM	PM

Description

The two low-order bits of the instruction word are copied into the product shift mode (PM) bits of ST1. The PM bits control the product register (PREG) output p-scaler shifter. The p-scaler shifter can shift the PREG output either 1 or 4 bits to the left or 6 bits to the right. The PM bit combinations and their meanings are shown below:

PM Field	Action
00	Output is not shifted
01	Output is left-shifted 1 bit and LSB is zero filled
10	Output is left-shifted 4 bits and 4 LSBs are zero filled
11	Output is right-shifted 6 bits, sign extended and 6 LSBs are lost

The left shifts allow the product to be justified for fractional arithmetic. The right shift by 6 accommodates up to 128 multiply-accumulate processes without overflow occurring. The PM bits may also be loaded by an LST #1 instruction (page 6-135).

SPM is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles The SPM instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Example

```
SPM 3 ;Product register shift mode 3 is selected, causing
      ;all subsequent transfers from the product register
      ;to the ALU to be shifted to the right six places.
```

Syntax

Direct: **SQRA** *dma*
Indirect: **SQRA** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(ACC) + (shifted PREG) → ACC
(dma) → TREG0
(dma) × (dma) → PREG

If TRM = 0:
(dma) → TREG1
(dma) → TREG2

Status Bits

Affected by:
OVM, PM, and TRM

Affects:
C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and added to the contents of the accumulator (ACC). The result is stored in the ACC. The contents of the data memory address (dma) are loaded into TREG0 and squared. The result is stored in PREG. The C bit is set, if the result of the addition generates a carry; otherwise, the C bit is cleared.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

SQRA is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
SQRA DAT30 ;(DP = 6, PM = 0)
```

Before Instruction

Data Memory		
31Eh		0Fh
TREG0		3h
PREG		12Ch
ACC	X	1F4h
	C	

After Instruction

Data Memory		
31Eh		0Fh
TREG0		0Fh
PREG		0E1h
ACC	0	320h
	C	

Example 2

```
SQRA *,AR4 ;(PM = 0)
```

Before Instruction

ARP		3
AR3		31Eh
Data Memory		
31Eh		0Fh
TREG0		3h
PREG		12Ch
ACC	X	1F4h
	C	

After Instruction

ARP		4
AR3		31Eh
Data Memory		
31Eh		0Fh
TREG0		0Fh
PREG		0E1h
ACC	0	320h
	C	

Syntax

Direct: **SQRS** *dma*
Indirect: **SQRS** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) – (shifted PREG) $\rightarrow$ ACC
(dma) $\rightarrow$ TREG0
(dma) $\times$ (dma) $\rightarrow$ PREG

If TRM = 0:
(dma) $\rightarrow$ TREG1
(dma) $\rightarrow$ TREG2

Status Bits

Affected by: OVM, PM, and TRM

Affects: C and OV

Description

The contents of the product register (PREG) are shifted, as defined by the PM bits, and subtracted from the contents of the accumulator (ACC). The result is stored in the ACC. The contents of the data memory address (dma) are loaded into TREG0 and squared. The result is stored in PREG. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs, maintaining ’C5x object-code compatibility with the ’C2x. The TREGs are memory-mapped registers and can be read and written with any instruction that accesses data memory. TREG1 has only 5 bits, and TREG2 has only 4 bits.

SQRS is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
SQRS DAT9 ;(DP = 6, PM = 0)
```

	Before Instruction		After Instruction	
Data Memory 309h		08h	Data Memory 309h	08h
TREG0		1124h	TREG0	08h
PREG		190h	PREG	40h
ACC	X	1450h	ACC	1	12C0h
	C			C

Example 2

```
SQRS *,AR5 ;(PM = 0)
```

	Before Instruction		After Instruction	
ARP		3	ARP	5
AR3		309h	AR3	309h
Data Memory 309h		08h	Data Memory 309h	08h
TREG0		1124h	TREG0	08h
PREG		190h	PREG	40h
ACC	X	1450h	ACC	1	12C0h
	C			C

Syntax

Direct: **SST** #*m*, *dma*
Indirect: **SST** #*m*, {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
m = 0 or 1
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing for SST#0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	0	dma						

Indirect addressing for SST#0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	1	See Section 5.2						

Direct addressing for SST#1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	0	dma						

Indirect addressing for SST#1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(STm) $\rightarrow$ dma

Status Bits

None affected.

Description

The contents of the status register STm are stored in the data memory address (dma). In the direct addressing mode, status register STm is always stored in data memory page 0, regardless of the value of the data memory page pointer (DP) bits. The processor automatically forces the data memory page to 0, and the specific location within that data page is defined by the instruction. The DP bits are not physically modified. This allows storage of the DP bits in the data memory on interrupts, etc., in the direct addressing mode without having to change the DP. In the indirect addressing mode, the dma is obtained from the selected auxiliary register (see the LST instruction, page 6-135, for more information). In the indirect addressing mode, any page in data memory may be accessed.

SST is a control instruction (see Table 6–10). Status registers ST0 and ST1 are defined in Section 4.4, *Status and Control Registers*.

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

Example 1

```
SST #0,DAT96 ;(DP = 6)
```

	Before Instruction		After Instruction
ST0	A408h	ST0	A408h
Data Memory 60h	0Ah	Data Memory 60h	A408h

Example 2

```
SST #1,*,AR7
```

	Before Instruction		After Instruction
ARP	0	ARP	7
AR0	300h	AR0	300h
ST1	2580h	ST1	2580h
Data Memory 300h	0h	Data Memory 300h	2580h

Syntax

Direct:	**SUB** *dma* [,*shift*]
Indirect:	**SUB** {*ind*} [,*shift* [,**AR***n*]]
Short immediate:	**SUB** #*k*
Long immediate:	**SUB** #*lk* [,*shift*]

Operands

$0 \le dma \le 127$
$0 \le shift \le 16$ (defaults to 0)
$0 \le n \le 7$
$0 \le k \le 255$
$-32768 \le lk \le 32767$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	SHFT †				0	dma						

† See Table 6–1 on page 6-2.

Indirect addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	SHFT †				1	See Section 5.2						

† See Table 6–1 on page 6-2.

Direct addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	1	0	dma						

Indirect addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0	8-Bit Constant							

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	1	0	SHFT †			
16-Bit Constant															

† See Table 6–1 on page 6-2.

Execution

Direct or indirect addressing:
$(PC) + 1 \rightarrow PC$
$(ACC) - ((dma) \times 2^{shift}) \rightarrow ACC$

Short immediate addressing:
(PC) + 1 $\rightarrow$ PC
(ACC) – k $\rightarrow$ ACC

Long immediate addressing:
(PC) + 2 $\rightarrow$ PC
(ACC) – (lk $\times$ 2^{shift}) $\rightarrow$ ACC

Status Bits

Affected by:	*Affects*:	
OVM and SXM	C and OV	Direct or indirect addressing
OVM	C and OV	Short immediate addressing
OVM and SXM	C and OV	Long immediate addressing

Description

If direct, indirect, or long immediate addressing is used, the contents of the data memory address (dma) or a 16-bit constant are shifted left, as defined by the shift code, and subtracted from the contents of the accumulator (ACC). The result is stored in the ACC. During shifting, the accumulator low byte (ACCL) is zero-filled. If the SXM bit is cleared, the high-order bits of the ACC are zero-filled; if the SXM bit is set, the high-order bits of the ACC are sign-extended.

Note that when the auxiliary register pointer (ARP) is updated during indirect addressing, you must specify a shift operand. If you don't want a shift, you must enter a 0 for this operand. For example:

```
SUB *+,0,AR0
```

If short immediate addressing is used, an 8-bit positive constant is subtracted from the contents of the ACC. The result is stored in the ACC. In this mode, no shift value may be specified and the subtraction is unaffected by the SXM bit.

The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set. If a 16-bit shift is specified with the SUB instruction, the C bit is cleared only if the result of the subtraction generates a borrow; otherwise, the C bit is unaffected.

SUB is an accumulator memory reference instruction (see Table 6–4).

Words

1 (Direct, indirect, or short immediate)

2 (Long immediate)

Cycles

For the short and long immediate addressing modes, the SUB instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

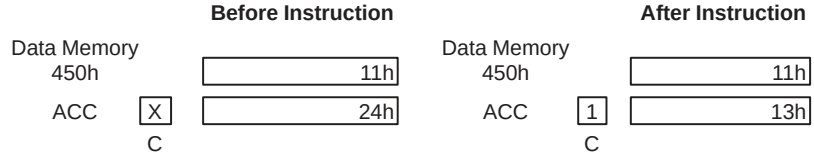
	ROM	DARAM	SARAM	External Memory
	1	1	1	1+p

Cycles for a Single Instruction (long immediate addressing)

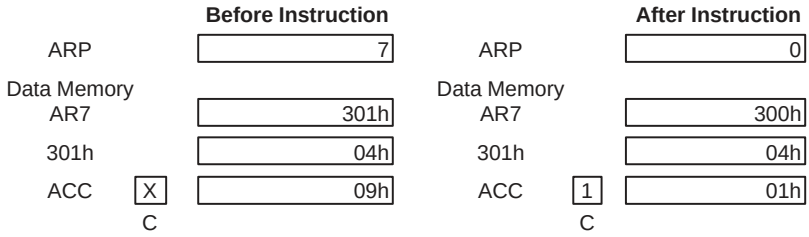
	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

Example 1

```
SUB DAT80 ;(DP = 8, SXM=0)
```



Example 2 `SUB *–,1,AR0 ;(SXM = 0)`



Example 3 `SUB #8h ;(SXM = 1)`

Example 4 `SUB #0FFFh,4 ;(SXM = 0)`

Syntax

Direct: **SUBB** *dma*
Indirect: **SUBB** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(ACC) – (dma) – (logical inversion of C) → ACC

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM	SXM	C and OV

Description

The contents of the data memory address (dma) and the logical inversion of the C bit are subtracted from the contents of the accumulator (ACC) with sign extension suppressed. The result is stored in the ACC. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

The SUBB instruction can be used in performing multiple-precision arithmetic. SUBB is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

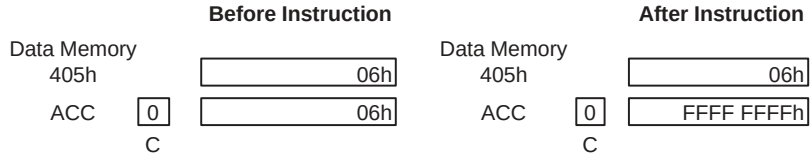
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

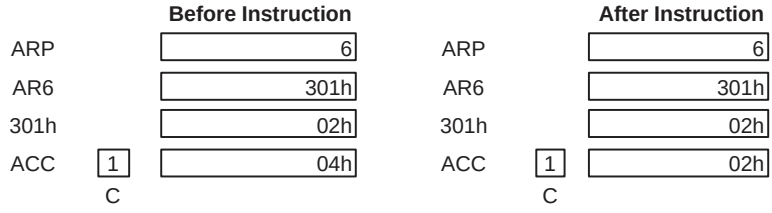
Example 1

```
SUBB DAT5 ;(DP = 8)
```



Example 2

```
SUBB *
```



In Example 1, the C bit is 0 from the result of a previous subtract instruction that performed a borrow. The operation performed was 6 – 6 – (1) = –1, generating another borrow (C = 0) in the process. In Example 2, no borrow was previously generated (C = 1), and the result from the subtract instruction does not generate a borrow.

Syntax

Direct: **SUBC** *dma*
Indirect: **SUBC** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) – ((dma) $\times$ 2^{15}) $\rightarrow$ ALU output

If ALU output $\ge$ 0:
    (ALU output) $\times$ 2 + 1 $\rightarrow$ ACC
Else:
    (ACC) $\times$ 2 $\rightarrow$ ACC

Status Bits

Not affected by:	*Affects*:
OVM (no saturation) and SXM	C and OV

Description

The SUBC instruction performs conditional subtraction, which may be used for division. The 16-bit dividend is stored in the accumulator low byte (ACCL) and the accumulator high byte (ACCH) is zero-filled. The divisor is in data memory. The SUBC instruction is executed 16 times for 16-bit division. After completion of the last SUBC instruction, the quotient of the division is in the ACCL and the remainder is in the ACCH. The SUBC instruction assumes that the divisor and the dividend are both positive. The divisor is not sign extended. The dividend, in the ACCL, must initially be positive (bit 31 must be 0) and must remain positive following the ACC shift, which occurs in the first portion of the SUBC execution.

If the 16-bit dividend contains fewer than 16 significant bits, the dividend may be placed in the ACC and shifted left by the number of leading nonsignificant zeroes. The number of SUBC executions is reduced from 16 by that number. One leading zero is always significant.

The SUBC instruction affects the OV bit, but is not affected by the OVM bit, and therefore the ACC does not saturate upon positive or negative overflows when executing this instruction. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

SUBC is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

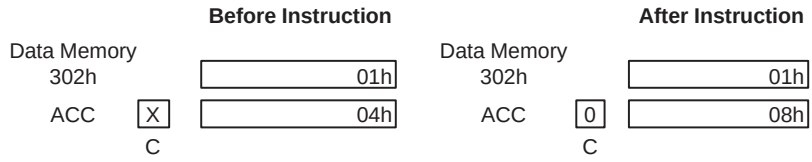
Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

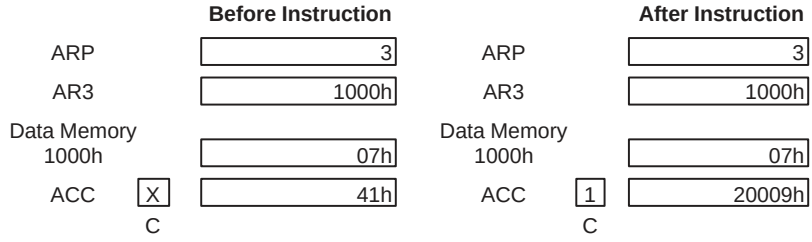
Example 1

```
SUBC DAT2 ;(DP = 6)
```



Example 2

```
RPT #15
SUBC *
```



Syntax

Direct: **SUBS** *dma*
Indirect: **SUBS** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	0	1	See Section 5.2						

Execution

(PC) + 1 $\rightarrow$ PC
(ACC) – (dma) $\rightarrow$ ACC
(dma) is an unsigned16-bit number

Status Bits

Affected by:	*Not affected by:*	*Affects*:
OVM	SXM	C and OV

Description

The contents of the data memory address (dma) are subtracted from the contents of the accumulator (ACC) with sign extension suppressed. The result is stored in the ACC. The data is treated as a 16-bit unsigned number, regardless of the SXM bit. The contents of the ACC are treated as a signed number. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

The SUBS instruction produces the same results as a SUB instruction (page 6-259) with the SXM bit cleared and a shift count of 0. SUBS is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
SUBS DAT2 ;(DP = 16, SXM = 1)
```

	Before Instruction		After Instruction	
Data Memory 802h		F003h		F003h
ACC	X	F105h	1	102h
	C		C	

Example 2

```
SUBS * ;(SXM = 1)
```

	Before Instruction		After Instruction	
ARP		0		0
AR0		310h		310h
Data Memory 310h		F003h		F003h
ACC	X	0FFF F105h	1	0FFF 0102h
	C		C	

Syntax

Direct: **SUBT** *dma*
Indirect: **SUBT** {ind} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	1	1	See Section 5.2						

Execution

$(PC) + 1 \rightarrow PC$
$(ACC) - ((dma) \times 2^{TREG1(3-0)}) \rightarrow (ACC)$

If SXM = 1:
(dma) is sign-extended
If SXM = 0:
(dma) is not sign-extended

Status Bits

Affected by:
OVM, SXM, and TRM

Affects:
C and OV

Description

The contents of the data memory address (dma) are shifted left from 0 to 15 bits, as defined by the 4 LSBs of TREG1, and subtracted from the contents of the accumulator (ACC). The result is stored in the ACC. Sign extension on the dma value is controlled by the SXM bit. The C bit is cleared, if the result of the subtraction generates a borrow; otherwise, the C bit is set.

You can maintain software compatibility with the ’C2x by clearing the TRM bit. This causes any ’C2x instruction that loads TREG0 to write to all three TREGs. Subsequent calls to the SUBT instruction will shift the value by the TREG1 value (which is the same as TREG0), maintaining ’C5x object-code compatibility with the ’C2x.

SUBT is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1

```
SUBT DAT127 ;(DP = 4)
```

	Before Instruction		After Instruction	
Data Memory 2FFh	06h		06h	
TREG1	08h		08h	
ACC	FDA5h	X	F7A5h	1
C				

Example 2

```
SUBT *
```

	Before Instruction		After Instruction	
ARP	1		1	
AR1	800h		800h	
Data Memory 800h	01h		01h	
TREG1	08h		08h	
ACC	0h	X	FFFF FF00h	0
C				

Syntax

Direct: **TBLR** *dma*
Indirect: **TBLR** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
(PFC) → MCS
(ACC(15–0)) → PFC

If (repeat counter) ≠ 0:
    (pma, addressed by PFC) → dma
    Modify current AR and ARP as specified
    (PFC) + 1 → PFC
    (repeat counter) –1 → repeat counter

Else:
    (pma, addressed by PFC) → dma
    Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

None affected.

Description

The contents of the program memory address (pma) are transferred to the data memory address (dma). The pma is specified by the contents of the accumulator low byte (ACCL) and the dma is specified by the instruction. A read from program memory is followed by a write to data memory to complete the instruction. When used with the RPT instruction, the TBLR instruction becomes a single-cycle instruction, once the RPT pipeline is started, and the program counter (PC) that contains the contents of the ACCL is incremented once each cycle.

TBLR is an I/O and data memory operation instruction (see Table 6–9).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	3	3	3	$3+p_{code}$
Source: SARAM Destination: DARAM	3	3	3	$3+p_{code}$
Source: External Destination: DARAM	$3+p_{src}$	$3+p_{src}$	$3+p_{src}$	$3+p_{src}+p_{code}$
Source: DARAM/ROM Destination: SARAM	3	3	3, 4†	$3+p_{code}$
Source: SARAM Destination: SARAM	3	3	3, 4†	$3+p_{code}$
Source: External Destination: SARAM	$3+p_{src}$	$3+p_{src}$	$3+p_{src}$, $4+p_{src}$†	$3+p_{src}+p_{code}$
Source: DARAM/ROM Destination: External	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+p_{code}$
Source: SARAM Destination: External	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+p_{code}$
Source: External Destination: External	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$6+p_{src}+d_{dst}+p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: SARAM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: External Destination: DARAM	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}+p_{code}$
Source: DARAM/ROM Destination: SARAM	n+2	n+2	n+2, n+4†	$n+2+p_{code}$

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, 2n+2§	n+2+p_{code}, 2n‡
Source: External Destination: SARAM	n+2+np_{src}	n+2+np_{src}	n+2+np_{src}, n+4+np_{src}†	n+2+np_{src}+p_{code}
Source: DARAM/ROM Destination: External	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+4+nd_{dst}+p_{code}
Source: SARAM Destination: External	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+4+nd_{dst}+p_{code}
Source: External Destination: External	4n+np_{src}+nd_{dst}	4n+np_{src}+nd_{dst}	4n+np_{src}+nd_{dst}	4n+2+np_{src}+nd_{dst} +p_{code}

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Example 1

```
TBLR DAT6 ;(DP = 4)
```

	Before Instruction		After Instruction
ACC	23h	ACC	23h
Program Memory 23h	306h	Program Memory 23h	306h
Data Memory 206h	75h	Data Memory 206h	306h

Example 2

```
TBLR *,AR7
```

	Before Instruction		After Instruction
ARP	0	ARP	7
AR0	300h	AR0	300h
ACC	24h	ACC	24h
Program Memory 24h	307h	Program Memory 24h	307h
Data Memory 300h	75h	Data Memory 300h	307h

Syntax

Direct: **TBLW** *dma*
Indirect: **TBLW** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	1	1	See Section 5.2						

Execution

(PC) + 1 → PC
(PFC) → MCS
(ACC(15–0)) → PFC

If (repeat counter) ≠ 0:
    (dma, addressed by PFC) → pma
    Modify current AR and ARP as specified
    (PFC) + 1 → PFC
    (repeat counter) –1 → repeat counter

    Else:
    (dma, addressed by PFC) → pma
    Modify current AR and ARP as specified
(MCS) → PFC

Status Bits

None affected.

Description

The contents of the data memory address (dma) are transferred to the program memory address (pma). The dma is specified by the instruction and the pma is specified by the contents of the accumulator low byte (ACCL). A read from data memory is followed by a write to program memory to complete the instruction. When used with the RPT instruction, the TBLW instruction becomes a single-cycle instruction, once the RPT pipeline is started, and the program counter (PC) that contains the contents of the ACCL is incremented once each cycle.

TBLW is an I/O and data memory operation instruction (see Table 6–9).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	3	3	3	$3+p_{code}$
Source: SARAM Destination: DARAM	3	3	3	$3+p_{code}$
Source: External Destination: DARAM	$3+d_{src}$	$3+d_{src}$	$3+d_{src}$	$3+d_{src}+p_{code}$
Source: DARAM Destination: SARAM	3	3	3, 4†	$3+p_{code}$
Source: SARAM Destination: SARAM	3	3	3, 4†	$3+p_{code}$
Source: External Destination: SARAM	$3+d_{src}$	$3+d_{src}$	$3+d_{src}$, $4+d_{src}$†	$3+d_{src}+p_{code}$
Source: DARAM Destination: External	$4+p_{dst}$	$4+p_{dst}$	$4+p_{dst}$	$5+p_{dst}+p_{code}$
Source: SARAM Destination: External	$4+p_{dst}$	$4+p_{dst}$	$4+p_{dst}$	$5+p_{dst}+p_{code}$
Source: External Destination: External	$4+d_{src}+p_{dst}$	$4+d_{src}+p_{dst}$	$4+d_{src}+p_{dst}$	$5+d_{src}+p_{dst}+p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: SARAM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: External Destination: DARAM	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}+p_{code}$
Source: DARAM Destination: SARAM	n+2	n+2	n+2, n+3†	$n+2+p_{code}$

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, 2n+1§	n+2+p_{code}, 2n‡
Source: External Destination: SARAM	n+2+nd_{src}	n+2+nd_{src}	n+2+nd_{src}, n+3+nd_{src}†	n+2+nd_{src}+p_{code}
Source: DARAM Destination: External	2n+2+np_{dst}	2n+2+np_{dst}	2n+2+np_{dst}	2n+3+np_{dst}+p_{code}
Source: SARAM Destination: External	2n+2+np_{dst}	2n+2+np_{dst}	2n+2+np_{dst}	2n+3+np_{dst}+p_{code}
Source: External Destination: External	4n+nd_{src}+np_{dst}	4n+nd_{src}+np_{dst}	4n+nd_{src}+np_{dst}	4n+1+nd_{src}+np_{dst} +p_{code}

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Example 1

```
TBLW DAT5 ;(DP = 32)
```

	Before Instruction		After Instruction
ACC	257h	ACC	257h
Data Memory 1005h	4339h	Data Memory 1005h	4339h
Program Memory 257h	306h	Program Memory 257h	4399h

Example 2

```
TBLW *
```

	Before Instruction		After Instruction
ARP	6	ARP	6
AR6	1006h	AR6	1006h
ACC	258h	ACC	258h
Data Memory 1006h	4340h	Data Memory 1006h	4340h
Program Memory 258h	307h	Program Memory 258h	4340h

Syntax **TRAP**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	0	0	0	1

Execution

(PC) + 1 → stack
22h → PC

Status Bits

Not affected by:
INTM

Does not affect:
INTM

Description

A software interrupt that transfers program control to program memory location 22h. The current program counter (PC) is incremented and pushed onto the stack. The address 22h is loaded into the PC. The instruction at address 22h may contain a branch instruction to transfer control to the TRAP routine. Placing the PC onto the stack enables a return instruction to pop the return address (pointing to the instruction after the TRAP) from the stack. The TRAP instruction is not maskable.

TRAP is a branch and call instruction (see Table 6–8).

Words 1

Cycles The TRAP instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

† The 'C5x performs speculative fetching by reading two additional instruction words. If PC discontinuity is taken, these two instruction words are discarded.

Example

```
TRAP ;Control is passed to program memory location 22h and
     ;PC + 1 is pushed onto the stack.
```

Syntax **XC** *n* ,*cond* [,*cond1*] [,...]

Operands n = 1 or 2

Conditions:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
TC = 0	NTC
$\overline{\text{TC}}$ = 1	TC
$\overline{\text{BIO}}$ low	BIO
Unconditional	UNC

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	N †	0	1	TP †		ZLVC †				ZLVC †			

† See Table 6–1 on page 6-2.

Operand (n) value	Opcode (N) value
1	0
2	1

Execution

If (condition(s)):
    next n instructions executed
Else:
    execute NOPs for next n instructions

Status Bits None affected.

Description If n = 1 and the conditions are met, the 1-word instruction following the XC instruction executes. If n = 2 and the conditions are met, the one 2-word instruction or two 1-word instructions following the XC instruction execute. Not all combinations of the conditions are meaningful. The XC instruction and the two instruction words following the XC are uninterruptible. If the conditions are not met, one or two NOPs are executed.

Conditions tested are sampled one full cycle before the XC is executed. Therefore, if the instruction prior to the XC is a single-cycle instruction, its execution will not affect the condition of the XC. If the instruction prior to the XC does affect the condition being tested, interrupt operation with the XC can cause undesired results.

XC is a branch and call instruction (see Table 6–8).

Words 1

Cycles The XC instruction is not repeatable.

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Example

```
XC 1,LEQ,C
MAR *+
ADD DAT100
```

If the contents of the accumulator are less than or equal to 0 and the C bit is set, the ARP is modified prior to the execution of the ADD instruction.

Syntax

Direct: **XOR** *dma*
Indirect: **XOR** {*ind*} [,**AR***n*]
Long immediate: **XOR** #*lk*, [,*shift*]

Operands

0 ≤ dma ≤ 127
0 ≤ n ≤ 7
lk: 16-bit constant
0 ≤ shift ≤ 16
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	1	See Section 5.2						

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	0	1	SHFT †			
16-Bit Constant															

† See Table 6–1 on page 6-2.

Long immediate addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	0	0	0	0	0	1	1
16-Bit Constant															

Execution

Direct or indirect addressing:
(PC) + 1 → PC
(ACC(15–0)) **XOR** (dma) → ACC(15–0)
(ACC(31–16)) → ACC(31–16)

Long immediate addressing:
(PC) + 2 → PC
(ACC(31–0)) **XOR** (lk × 2^{shift}) → ACC(31–0)

Status Bits

Does not affect: C
Not affected by: SXM Long immediate addressing

Description

If a long immediate constant is specified, the constant is shifted left, as defined by the shift code, and zero-extended on both ends and is exclusive-ORed with the contents of the accumulator (ACC). The result is stored in the ACC. If a constant is not specified, the contents of the data memory address (dma) are exclusive-ORed with the contents of the accumulator low byte (ACCL). The result is stored in the ACCL and the contents of the accumulator high byte (ACCH) are unaffected.

XOR is an accumulator memory reference instruction (see Table 6–4).

Words

1 (Direct or indirect addressing)

2 (Long immediate addressing)

Cycles

For the long immediate addressing modes, the XOR instruction is not repeatable.

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

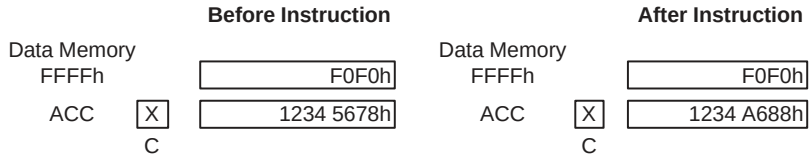
Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

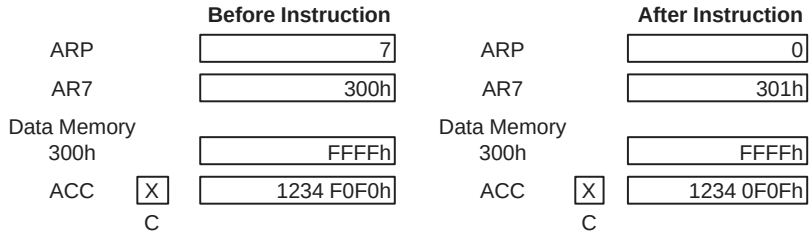
Cycles for a Single Instruction (long immediate addressing)

	ROM	DARAM	SARAM	External Memory
	2	2	2	2+2p

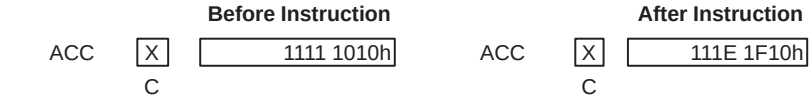
Example 1 `XOR DAT127 ;(DP = 511)`



Example 2 `XOR *+,AR0`



Example 3 `XOR #0F0F0h,4`



Syntax **XORB**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	1	0

Execution

(PC) + 1 → PC
(ACC) **XOR** (ACCB) → ACC

Status Bits None affected.

Description The contents of the accumulator (ACC) are exclusive-ORed with the contents of the accumulator buffer (ACCB). The result is stored in the ACC and the contents of the ACCB are unaffected.

XORB is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example X0RB

	Before Instruction		After Instruction
ACCB	F0F0 F0F0h	ACCB	F0F0 F0F0h
ACC	FFFF 0000h	ACC	0F0F F0F0h

Syntax

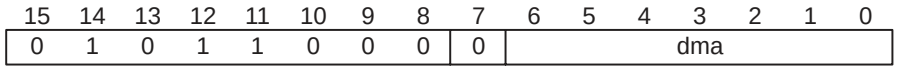
Direct: **XPL** [#*lk*,] *dma*
Indirect: **XPL** [#*lk*,] {*ind*} [,**AR***n*]

Operands

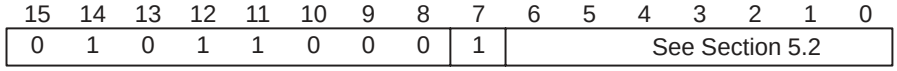
0 ≤ dma ≤ 127
lk: 16-bit constant
0 ≤ n ≤ 7
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

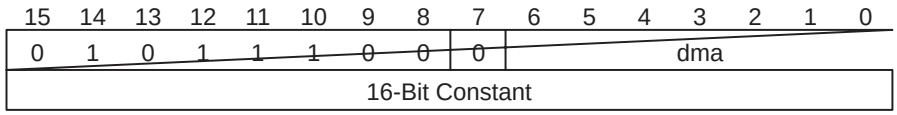
Direct addressing with long immediate not specified



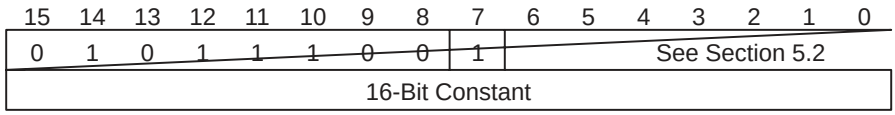
Indirect addressing with long immediate not specified



Direct addressing with long immediate specified



Indirect addressing with long immediate specified



Execution

Long immediate not specified:
(PC) + 1 → PC
(dma) **XOR** (DBMR) → dma

Long immediate specified:
(PC) + 2 → PC
(dma) **XOR** lk → dma

Status Bits

Affects: TC

Description

If a long immediate constant is specified, the constant is exclusive-ORed with the contents of the data memory address (dma). If a constant is not specified, the contents of the dma are exclusive-ORed with the contents of the dynamic bit manipulation register (DBMR). In both cases, the result is written directly back to the dma. The contents of the accumulator (ACC) are unaffected. If the result of the XOR operation is 0, the TC bit is set; otherwise, the TC bit is cleared.

XPL is a parallel logic unit (PLU) instruction (see Table 6–6).

Words

1 (Long immediate not specified)

2 (Long immediate specified)

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n–2	2n–2	2n–2, 2n+1†	2n–2+p
External	4n–2+2nd	4n–2+2nd	4n–2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2	2+2p
External	3+2d	3+2d	3+2d	6+2d+2p

Cycles for a Repeat (RPT) Execution (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	2n–1	2n–1	2n–1, 2n+2†	2n–1+2p
External	4n–1+2nd	4n–1+2nd	4n–1+2nd	4n+2+2nd+2p

† If the operand and the code are in the same SARAM block

Example 1

```
XPL #100h,DAT60 ;(DP = 0)
```

	Before Instruction		After Instruction
Data Memory 60h	01h	Data Memory 60h	101h

Example 2

```
XPL DAT60 ;(DP=0)
```

	Before Instruction		After Instruction
DBMR	FFFFh	DBMR	FFFFh
Data Memory 60h	0101h	Data Memory 60h	FEFEh

Example 3

```
XPL #1000h,*,AR6
```

	Before Instruction		After Instruction
ARP	0	ARP	6
AR0	300h	AR0	300h
Data Memory 300h	FF00h	Data Memory 300h	EF00h

Example 4

```
XPL *–,AR0
```

	Before Instruction		After Instruction
ARP	6	ARP	0
AR6	301h	AR6	300h
DBMR	FF00h	DBMR	FF00h
Data Memory 301h	EF00h	Data Memory 301h	1000h

Syntax

Direct: **ZALR** *dma*
Indirect: **ZALR** {*ind*} [,**AR***n*]

Operands

$0 \le dma \le 127$
$0 \le n \le 7$
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Opcode

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	1	See Section 5.2						

Execution

(PC) + 1 → PC
8000h → ACC(15–0)
(dma) → ACC(31–16)

Status Bits

Does not affect: C

Description

The contents of the data memory address (dma) are loaded into the accumulator high byte (ACCH). The ZALR instruction rounds the value by adding 1/2 LSB; that is, the 15 low-order bits (bits 0–14) of the accumulator low byte (ACCL) are cleared, and ACCL bit 15 is set.

ZALR is an accumulator memory reference instruction (see Table 6–4).

Words

1

Cycles

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Example 1 `ZALR DAT3 ;(DP = 32)`

	Before Instruction	After Instruction
Data Memory 1003h	3F01h	3F01h
ACC	0077 FFFFh	3F01 8000h
C	X	X

Example 2 `ZALR *-,AR4`

	Before Instruction	After Instruction
ARP	7	4
AR7	FF00h	FEFFh
Data Memory FF00h	E0E0h	E0E0h
ACC	0010 7777h	E0E0 8000h
C	X	X

Syntax **ZAP**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	0	1

Execution

(PC) + 1 → PC
0 → ACC
0 → PREG

Status Bits None affected.

Description The contents of the accumulator (ACC) and product register (PREG) are cleared. The ZAP instruction speeds up the preparation for a repeat multiply/accumulate.

ZAP is an accumulator memory reference instruction (see Table 6–4).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example ZAP

	Before Instruction		After Instruction
PREG	3F01 1111h	PREG	0000 0000h
ACC	77FF FF77h	ACC	0000 0000h

Syntax **ZPR**

Operands None

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	0	0

Execution

(PC) + 1 → PC
0 → PREG

Status Bits None affected.

Description The contents of the product register (PREG) are cleared. ZPR is a TREG0, PREG, and multiply instruction (see Table 6–7).

Words 1

Cycles

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

Example ZPR

	Before Instruction		After Instruction
PREG	3F01 1111h	PREG	0000 0000h

Chapter 7

Pipeline

In the operation of the pipeline, the instruction fetch, decode, operand read, and execute operations are independent, which allows overall instruction executions to overlap.

Topic **Page**

7.1 Pipeline Structure

The four phases of the ’C5x pipeline structure and their functions are as follows:

1) Fetch (F) — This phase fetches the instruction words from memory and updates the program counter (PC).

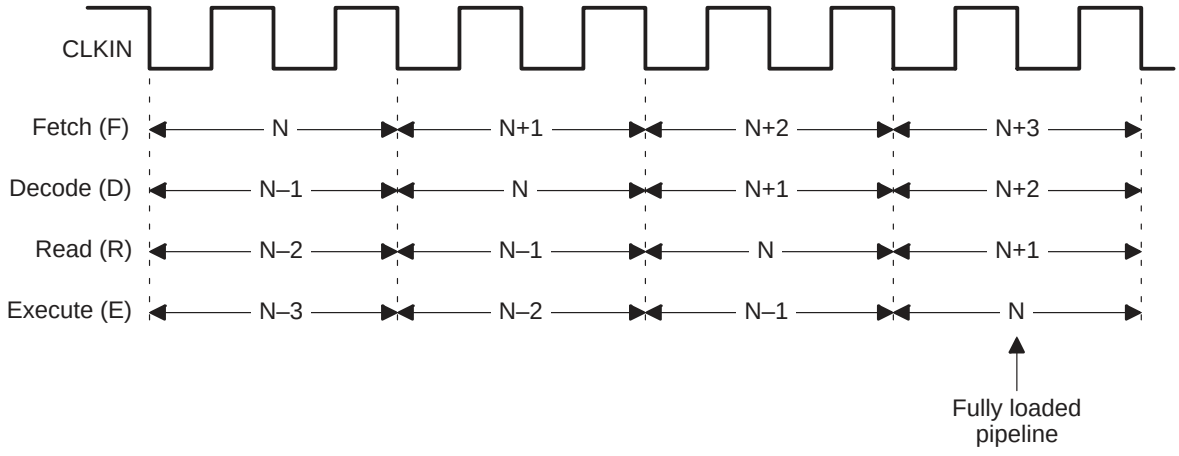
2) Decode (D) — This phase decodes the instruction word and performs address generation and ARAU updates of auxiliary registers.

3) Read (R) — This phase reads operands from memory, if required. If the instruction uses indirect addressing mode, it will read the memory location pointed at by the ARP before the update of the previous decode phase.

4) Execute (E) — This phase performs any specify operation, and, if required, writes results of a previous operation to memory.

Figure 7–1 illustrates the operation of the four-level pipeline for single-word single-cycle instruction executing with no wait state. This is perfect overlapping in the pipeline, where all four phases operate in parallel. When more than one pipeline stage requires processing on the same resource, such as memory and CPU registers, a pipeline conflict occurs. Since there is no priority between these four phases, you can get unexpected results when pipeline conflict occurs. Therefore, you should avoid any conflict between these four phases in order to get the correct results.

Figure 7–1. Four Level Pipeline Operation



7.2 Pipeline Operation

The pipeline is essentially invisible to the user except in some cases, such as auxiliary register updates, memory-mapped accesses of the CPU registers, the NORM instruction, and memory configuration commands. Furthermore, the pipeline operation is not protected. The user has to understand the pipeline operation to avoid the pipeline conflict by arranging the code. The following sections show how the pipeline operation and how the pipeline conflict affect the result.

7.2.1 Normal Pipeline Operation

Example 7–1 shows the pipeline operation of a 1-word instruction and Example 7–2 shows the pipeline operation of a 2-word instruction.

1-Word Instruction

Example 7–1. Pipeline Operation of 1-Word Instruction

```
ADD    *+
SAMM   TREG0
MPY    *+
SQRA   *+, AR2
.
.
.
```

Table 7–1. Pipeline Operation of 1-Word Instruction

		Pipeline operation								
Cycle	PC	F	D	R	E	ARP	AR6	TREG0	PREG	ACC
1	[SAMM]	ADD				6	60h	XX	XX	20h
2	[MPY]	SAMM	ADD			6	61h	XX	XX	20h
3	[SQRA]	MPY	SAMM	ADD		6	61h	XX	XX	20h
4		SQRA	MPY	SAMM	ADD	6	62h	XX	XX	30h
5			SQRA	MPY	SAMM	2	63h	30h	XX	30h
6				SQRA	MPY	X	XX	30h	90h	30h
7					SQRA	X	XX	06h	24h	C0h

Assume memory locations 60h = 10h, 61h = 3h, and 62h = 6h. The following is the condition of the pipeline for each cycle.

Cycle 1:
- F) Fetch the ADD instruction and update PC to next instruction.

Cycle 2:
- F) Fetch the SAMM instruction and update PC.
- D) Decode the ADD instruction, generate address, and update AR6.

Cycle 3:
- F) Fetch the MPY instruction and update PC.
- D) Decode the SAMM instruction, no address generate, and no ARAU update.
- R) Read data from memory location 60h (10h) which is the location pointed at by AR6 before the update of cycle 2.

Cycle 4:
- F) Fetch the SQRA instruction and update PC.
- D) Decode the MPY instruction and update AR6.
- R) No operand read for the SAMM instruction.
- E) Add data read in cycle 3 (10h) to data in ACC (20h) and store result in ACC (ACC = 30h).

Cycle 5:
- F) Fetch the next instruction and update PC.
- D) Decode the SQRA instruction, and update AR6 and ARP.
- R) Read data from data memory location 61h (3h) which is the location pointed at by AR6 before the update of cycle 4.
- E) Store data in ACC to TREG0 (TREG0 = 30h).

Cycle 6:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 5.
- R) Read data from data memory location 62h (6h) which is the location pointed at by AR6 before the update of cycle 5.
- E) Multiply data in TREG0 (30h) with data read in cycle 5 (3h) and store result in PREG (PREG = 90h).

Cycle 7:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 6.
- R) Depends on the instruction fetched in cycle 5.
- E) Add data in ACC (30h) to data in PREG (90h) and store result in ACC (ACC = C0h). Store data read in cycle 6 (6h) to TREG0. Square data in TREG0 (6h) and store result in PREG (PREG = 24h).

2-Word Instruction

Example 7–2. Pipeline Operation of 2-Word Instruction

```
LACC   *+
ADD    #1000h
SACL   *+, 0, AR2
.
.
.
```

Table 7–2. Pipeline Operation of 2-Word Instruction

		Pipeline operation							
Cycle	**PC**	**F**	**D**	**R**	**E**	**ARP**	**AR1**	**ACC**	**[61]**
1	[ADD]	LACC				1	60h	20h	3h
2	[1000h]	ADD	LACC			1	61h	20h	3h
3	[SACL]	1000h	ADD	LACC		1	61h	20h	3h
4		SACL	dummy	ADD	LACC	1	61h	10h	3h
5			SACL	dummy	ADD	2	62h	1010h	3h
6				SACL	dummy	X	XX	1010h	3h
7					SACL	X	XX	1010h	1010h

Assume memory location 60h = 10h and 61h = 3h. The following is the condition of the pipeline for each cycle.

Cycle 1: F) Fetch the LACC instruction and update PC to next instruction.

Cycle 2: F) Fetch the ADD instruction and update PC.

D) Decode the LACC instruction and update AR1.

Cycle 3: F) Fetch the second word 1000h and update PC.

D) Decode the ADD instruction and no ARAU update.

R) Read data from data memory location 60h (10h) which is the location pointed at by AR1 before the update of cycle 2.

Cycle 4: F) Fetch the SACL instruction and update PC.

D) Dummy operation (previous fetch phase is an operand).

R) No operand read for the ADD instruction.

E) Load ACC with data read in cycle 3 (ACC = 10h).

Cycle 5: F) Fetch the next instruction and update PC.

D) Decode the SACL instruction, and update AR1 and ARP.

R) Dummy operation (operand fetch on fetch phase).

E) Add 1000h to data in ACC (10h) and store result in ACC (ACC = 1010h).

Cycle 6: F) Fetch the next instruction and update PC.

D) Decode the instruction fetched in cycle 5.

R) No operand read for the SACL instruction.

E) Dummy operation (operand fetch on fetch phase).

Cycle 7: F) Fetch the next instruction and update PC.

D) Decode the instruction fetched in cycle 6.

R) Depends on the instruction fetched in cycle 5.

E) Store data in ACC (1010h) to data memory location 61h which is the location pointed at by AR1 before the update of cycle 5.

7.2.2 Pipeline Operation on Branch and Subroutine Call

Since the pipeline is 4-levels deep, normally any branch, subroutine call, or return from subroutine instruction (Table 6–8 on page 6-17) takes 4 cycles to flush the pipeline. The conditional branch (BCND) instruction also takes 4 cycles, when the condition is true. Following are examples that show the pipeline operations of the conditional branch, subroutine call, and return from subroutine instructions.

Branch Taken

Example 7–3. Pipeline Operation with Branch Taken

```
ADD    *+
BCND   LBL, NEQ      ; Branch if ACC ≠ 0
ADD    *+
SUB    #1
SACL   *+,0,AR2
LBL
SUB    *+
.
.
.
```

Table 7–3. Pipeline Operation with Branch Taken

Cycle	PC	Pipeline operation				ARP	AR1	ACC
		F	D	R	E			
1	[BCND]	ADD				1	60h	20h
2	[LBL]	BCND	ADD			1	61h	20h
3	[ADD]	LBL	BCND	ADD		1	61h	20h
4	[SUB]	ADD	dummy	BCND	ADD	1	61h	30h
5	LBL	SUB	dummy	dummy	BCND	1	61h	30h
6		SUB	dummy	dummy	dummy	1	61h	30h
7			SUB	dummy	dummy	1	62h	30h
8				SUB	dummy	X	XX	30h
9					SUB	X	XX	2Dh

Assume memory location 60h = 10h and 61h = 3h. The following is the condition of the pipeline for each cycle.

Cycle 1: F) Fetch the ADD instruction and update PC to next instruction.

Cycle 2: F) Fetch the BCND instruction and update PC.

D) Decode the ADD instruction and update AR1.

Cycle 3: F) Fetch the second word LBL and update PC.

D) Decode the BCND instruction and no ARAU update.

R) Read data from data memory location 60h (10h) which is the location pointed at by AR1 before the update of cycle 2.

Cycle 4: F) Fetch the ADD instruction and update PC.

D) Dummy operation (previous fetch phase is an operand).

R) No operand read for the BCND instruction.

E) Add data read in cycle 3 (10h) to data in ACC (20h) and store result in ACC (ACC = 30h).

The PC update and decode (D) phase on cycle 5 depends on the execute (E) phase result of the BCND instruction. Since the condition is true, the PC will update to point to the destination address and a dummy operation will be inserted in the decode (D) phase to flush the pipeline.

Cycle 5: F) Fetch the SUB instruction and update PC. Since the condition is true, the operand of BCND (LBL) will copy to PC.

D) Dummy operation (flush the pipeline).

R) Dummy operation (operand fetch on fetch phase).

E) Conditional testing.

Cycle 6: F) Fetch the SUB instruction and update PC.

D) Dummy operation (flush the pipeline).

R) Dummy operation (flush the pipeline).

E) Dummy operation (operand fetch on fetch phase).

Cycle 7: F) Fetch the next instruction and update PC.

D) Decode the SUB instruction and update AR1.

R) Dummy operation (flush the pipeline).

E) Dummy operation (flush the pipeline).

Cycle 8: F) Fetch the next instruction and update PC.

D) Decode the instruction in cycle 7.

R) Read data from data memory location 61h (3h) which is the location pointed at by AR1 before the update of cycle 7.

E) Dummy operation (flush the pipeline).

Cycle 9: F) Fetch the next instruction and update PC.

D) Decode the instruction fetched in cycle 8.

R) Depends on the instruction fetched in cycle 7.

E) Subtract data read in cycle 8 (3h) from data in ACC (30h) and store result in ACC (ACC = 2Dh).

Branch Not Taken

Example 7–4. Pipeline Operation with Branch Not Taken

```
ADD    *+
BCND   LBL, EQ       ;Branch if ACC = 0
ADD    *+
SUB    #1
SACL   *+,0,AR2
.
.
LBL
SUB    *+
.
.
```

Table 7–4. Pipeline Operation with Branch Not Taken

Cycle	PC	Pipeline operation				ARP	AR1	ACC	[62h]
		F	D	R	E				
1	[BCND]	ADD				1	60h	20h	9h
2	[LBL]	BCND	ADD			1	61h	20h	9h
3	[ADD]	LBL	BCND	ADD		1	61h	20h	9h
4	[SUB]	ADD	dummy	BCND	ADD	1	61h	30h	9h
5	SACL	SUB	ADD	dummy	BCND	1	62h	30h	9h
6		SACL	SUB	ADD	dummy	1	62h	30h	9h
7			SACL	SUB	ADD	2	63h	33h	9h
8				SACL	SUB	X	XX	32h	9h
9					SACL	X	XX	32h	32h

Assume memory location 60h = 10h, 61h = 3h, and 62h = 9h. The following is the condition of the pipeline for each cycle.

Cycle 1: F) Fetch the ADD instruction and update PC to next instruction.

Cycle 2: F) Fetch the BCND instruction and update PC.

D) Decode the ADD instruction and update AR1.

Cycle 3: F) Fetch the second word LBL and update PC.

D) Decode the BCND instruction and no ARAU update.

R) Read data from data memory location 60h (10h) which is the location pointed at by AR1 before the update of cycle 2.

Cycle 4: F) Fetch the ADD instruction and update PC.

D) Dummy operation (previous fetch phase is an operand).

R) No operand read for the BCND instruction.

E) Add data read in cycle 3 (10h) to data in ACC (20h) and store result in ACC (ACC = 30h).

The PC update and decode (D) phase on cycle 5 depends on the execute (E) phase result of the BCND instruction. Since the condition is false, the PC will update to point to the next instruction and BCND will be treated as 2-word instruction.

Cycle 5: F) Fetch the SUB instruction and update PC.

D) Decode the ADD instruction and update AR1.

R) Dummy operation (operand fetch on fetch phase).

E) Conditional testing.

Cycle 6: F) Fetch the SACL instruction and update PC.

D) Decode the SUB instruction and no ARAU update.

R) Read data from data memory location 61h (3h) which is the location pointed at by AR1 before the update of cycle 5.

E) Dummy operation (operand fetch on fetch phase).

Cycle 7: F) Fetch the next instruction and update PC.

D) Decode the SACL instruction, and update AR1 and ARP.

R) No operand read for the SUB instruction.

E) Add data read in cycle 6 (3h) to data in ACC (30h) and store result in ACC (ACC = 33h).

Cycle 8: F) Fetch the next instruction and update PC.

D) Decode the instruction fetched in cycle 7.

R) No operand read for the SACL instruction.

E) Subtract 1h from data in ACC (33h) and store result in ACC (ACC = 32h).

Cycle 9: F) Fetch the next instruction and update PC.

D) Decode the instruction fetched in cycle 8.

R) Depends on the instruction fetched in cycle 7.

E) Store data in ACC (32h) to data memory location 62h which is the location pointed at by AR1 before the update of cycle 7.

Subroutine Call and Return

Example 7–5. Pipeline Operation with Subroutine Call and Return

```
ADD    *+
CALL   LBL
ADD    *+
SUB    #1
SACL   *+,0,AR2
.
.
LBL
SUBB   *+
RET
NOP
NOP
NOP
.
.
```

Table 7–5. Pipeline Operation with Subroutine Call and Return

		Pipeline operation								
Cycle	**PC**	**F**	**D**	**R**	**E**	**ARP**	**AR1**	**ACC**	**TOS**	**[63h]**
1	[CALL]	ADD				1	60h	20h	XX	XX
2	[LBL]	CALL	ADD			1	61h	20h	XX	XX
3	[ADD]	LBL	CALL	ADD		1	61h	20h	XX	XX
4	[SUB]	ADD	dummy	CALL	ADD	1	61h	30h	XX	XX
5	LBL	SUB	dummy	dummy	CALL	1	61h	30h	[ADD]	XX
6	[RET]	SUBB	dummy	dummy	dummy	1	61h	30h	[ADD]	XX
7	[NOP]	RET	SUBB	dummy	dummy	1	62h	30h	[ADD]	XX
8	[NOP]	NOP	RET	SUBB	dummy	1	62h	30h	[ADD]	XX
9	[NOP]	NOP	dummy	RET	SUBB	1	62h	2Dh	[ADD]	XX
10	[ADD]	NOP	dummy	dummy	RET	1	62h	2Dh	XX	XX
11	[SUB]	ADD	dummy	dummy	dummy	1	62h	2Dh	XX	XX
12	[SACL]	SUB	ADD	dummy	dummy	1	63h	2Dh	XX	XX
13		SACL	SUB	ADD	dummy	1	63h	2Dh	XX	XX
14			SACL	SUB	ADD	2	64h	36h	XX	XX
15				SACL	SUB	XX	XX	35h	XX	XX
16					SACL	XX	XX	35h	XX	35h

Assume memory location 60h = 10h, 61h = 3h, and 62h = 9h. The following is the condition of the pipeline for each cycle.

Cycle 1:
- F) Fetch the ADD instruction and update PC to next instruction.

Cycle 2:
- F) Fetch the CALL instruction and update PC.
- D) Decode the ADD instruction and update AR1.

Cycle 3:
- F) Fetch the second word LBL and update PC.
- D) Decode the CALL instruction and no ARAU update.
- R) Read data from data memory location 60h (10h) which is the location pointed at by AR1 before the update of cycle 2.

Cycle 4:
- F) Fetch the ADD instruction and update PC.
- D) Dummy operation (previous fetch phase is an operand).
- R) No operand read for the CALL instruction.
- E) Add data read in cycle 3 (10h) to data in ACC (20h) and store result in ACC (ACC = 30h).

Cycle 5:
- F) Fetch the SUB instruction. PC will modify during the execution (E) phase.
- D) Dummy operation (flush the pipeline).
- R) Dummy operation (operand fetch on fetch phase).
- E) Push the address of ADD on top of stack (TOS). Update PC equal to LBL (ready to enter the subroutine).

Cycle 6:
- F) Fetch the SUBB instruction and update PC.
- D) Dummy operation (flush the pipeline).
- R) Dummy operation (flush the pipeline).
- E) Dummy operation (operand fetch on fetch phase).

Cycle 7:
- F) Fetch the RET instruction and update PC.
- D) Decode the SUBB instruction and update AR1.
- R) Dummy operation (flush the pipeline).
- E) Dummy operation (flush the pipeline).

Cycle 8:
- F) Fetch the NOP instruction and update PC.
- D) Decode the RET instruction and no ARAU update.
- R) Read data from data memory location 61h (3h) which is the location pointed at by AR1 before the update of cycle 7.
- E) Dummy operation (flush the pipeline).

Cycle 9: F) Fetch the NOP instruction and update PC.
D) Dummy operation (flush the pipeline).
R) No operand read for the RET instruction.
E) Subtract data read in cycle 8 (3h) from data in ACC (30h) and store result in ACC (ACC = 2Dh).

Cycle 10: F) Fetch the NOP instruction. PC will modify during the execute (E) phase.
D) Dummy operation (flush the pipeline).
R) Dummy operation (flush the pipeline).
E) Pop the address from TOS to PC (ready to return from subroutine).

Cycle 11: F) Fetch the ADD instruction and update PC.
D) Dummy operation (flush the pipeline).
R) Dummy operation (flush the pipeline).
E) Dummy operation (flush the pipeline).

Cycle 12: F) Fetch the SUB instruction and update PC.
D) Decode the ADD instruction and update AR1.
R) Dummy operation (flush the pipeline).
E) Dummy operation (flush the pipeline).

Cycle 13: F) Fetch the SACL instruction and update PC.
D) Decode the SUB instruction and no ARAU update.
R) Read data from data memory location 62h (9h) which is the location pointed at by AR1 before the update of cycle 12.
E) Dummy operation (flush the pipeline).

Cycle 14: F) Fetch the next instruction and update PC.
D) Decode the SACL instruction, and update AR1 and ARP.
R) No operand read for the SUB instruction.
E) Add data read in cycle 13 (9h) to data in ACC (2Dh) and store result in ACC (ACC = 36h).

Cycle 15: F) Fetch the next instruction and update PC.
D) Decode the instruction fetched in cycle 14.
R) No operand read for the SACL instruction.
E) Subtract 1h from data in ACC (36h) and store result in ACC (ACC = 35h).

Cycle 16: F) Fetch the next instruction and update PC.

D) Decode the instruction fetched in cycle 15.

R) Depends on the instruction fetched in cycle 14.

E) Store data in ACC (35h) to data memory location 63h which is the location pointed at by AR1 before the update of cycle 14.

7.2.3 Pipeline Operation on ARAU Memory-Mapped Registers

Auxiliary register arithmetic unit (ARAU) updates of the ARs occur during the decode (D) phase of the pipeline. This allows the address to be generated before the operand read (R) phase. However, memory-mapped accesses (for example, SAMM, LMMR, SACL, or SPLK) to the ARs occur in the execute (E) phase of the pipeline. Therefore, the use of ARs for the next two instructions after a memory-mapped load of the AR is prohibited. This means that the next two instructions after a memory-mapped load of the AR should not use this AR.

Modifications to the index register (INDX) and auxiliary register compare register (ARCR) also occur in the execute (E) phase of the pipeline. Therefore, any AR updates using the INDX or the ARCR must take place at least two cycles after a load of these registers. Example 7–6, Example 7–7, and Example 7–8 show the effects of a memory-mapped write to an auxiliary register.

Example 7–6. Pipeline Operation with ARx Load

```
LAR    AR2,#67h
LACC #64h
SAMM   AR2
LACC   *–
ADD    *–
.
.
```

Table 7–6. Pipeline Operation with ARx Load

Cycle	PC	Pipeline operation				ARP	AR2	ACC
		F	D	R	E			
1	[LACC]	LAR				2	XX	XX
2	[#64h]	LACC	LAR			2	XX	XX
3	[SAMM]	64h	LACC	LAR		2	XX	XX
4	[LACC]	SAMM	dummy	LACC	LAR	2	67h	XX
5	[ADD]	LACC	SAMM	dummy	LACC	2	67h	64h
6		ADD	LACC	SAMM	dummy	2	66h	64h
7			ADD	LACC	SAMM	2	64h	64h
8				ADD	LACC	2	64h	50h
9					ADD	2	64h	90h

Assume memory location 65h = 30h, 66h = 40h, and 67h = 50h. The following is the condition of the pipeline for each cycle.

Cycle 1: F) Fetch the LAR instruction and update PC to next instruction.

Cycle 2: F) Fetch the LACC instruction and update PC.
D) Decode the LAR instruction and no ARAU update.

Cycle 3: F) Fetch the second word 64h and update PC.
D) Decode the LACC instruction and no ARAU update.
R) No operand read for the LAR instruction.

Cycle 4: F) Fetch the SAMM instruction and update PC.
D) Dummy operation (previous fetch phase is an operand).
R) No operand read for the LACC instruction.
E) Load AR2 with 67h.

Cycle 5: F) Fetch the LACC instruction and update PC.
D) Decode the SAMM instruction and no ARAU update.
R) Dummy operation (operand fetch on fetch phase).
E) Load ACC with 64h.

Cycle 6: F) Fetch the ADD instruction and update PC.
D) Decode the LACC instruction and update AR2.
R) No operand read for the SAMM instruction.
E) Dummy operation (operand fetch on fetch phase).

Cycle 7:
- F) Fetch the next instruction and update PC.
- D) Decode the ADD instruction and update AR2.
- R) Read data from data memory location 67h (50h) which is the location pointed at by AR2 before the update of cycle 6.
- E) Store data in ACC (64h) to AR2. This conflicts with decode (D) phase.

Cycle 8:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 7.
- R) Read data from data memory location 66h (40h) which is the location pointed at by AR2 before the update of cycle 7.
- E) Load ACC with data read in cycle 7 (ACC = 50h).

Cycle 9:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 8.
- R) Depends on the instruction fetched in cycle 7.
- E) Add data read in cycle 8 (40h) to data in ACC (50h) and store result in ACC (ACC = 90h).

Example 7–7. Pipeline Operation with ARx Load and NOP Instruction

```
LAR    AR2,#67h
LACC #64h
SAMM   AR2
LACC   *–
NOP
ADD    *–
.
.
```

Table 7–7. Pipeline Operation with ARx Load and NOP Instruction

Cycle	PC	Pipeline operation				ARP	AR2	ACC
		F	D	R	E			
1	[LACC]	LAR				2	XX	XX
2	[#64h]	LACC	LAR			2	XX	XX
3	[SAMM]	64h	LACC	LAR		2	XX	XX
4	[LACC]	SAMM	dummy	LACC	LAR	2	67h	XX
5	[NOP]	LACC	SAMM	dummy	LACC	2	67h	64h
6	[ADD]	NOP	LACC	SAMM	dummy	2	66h	64h
7		ADD	dummy	LACC	SAMM	2	64h	64h
8			ADD	dummy	LACC	2	63h	64h
9				ADD	dummy	2	63h	50h
10					ADD	2	63h	70h

Assume memory location 63h = 10h, 64h = 20h, 65h = 30h, 66h = 40h, and 67h = 50h. The following is the condition of the pipeline for each cycle.

Cycle 1: F) Fetch the LAR instruction and update PC to next instruction.

Cycle 2: F) Fetch the LACC instruction and update PC.

D) Decode the LAR instruction and no ARAU update.

Cycle 3: F) Fetch the second word 64h and update PC.

D) Decode the LACC instruction and no ARAU update.

R) No operand read for the LAR instruction.

Cycle 4: F) Fetch the SAMM instruction and update PC.

D) Dummy operation (previous fetch (F) phase is an operand).

R) No operand read for the LACC instruction.

E) Load AR2 with 67h.

Cycle 5: F) Fetch the LACC instruction and update PC.

D) Decode the SAMM instruction and no ARAU update.

R) Dummy operation (operand fetch on fetch phase).

E) Load ACC with 64h.

Cycle 6:
- F) Fetch the NOP instruction and update PC.
- D) Decode the LACC instruction and update AR2.
- R) No operand read for the SAMM instruction.
- E) Dummy operation (operand fetch on fetch phase).

Cycle 7:
- F) Fetch the ADD instruction and update PC.
- D) Dummy operation (flush the pipeline).
- R) Read data from data memory location 67h (50h) which is the location pointed at by AR2 before the update of cycle 6.
- E) Store data in ACC to AR2 (AR2 = 64h).

Cycle 8:
- F) Fetch the next instruction and update PC.
- D) Decode the ADD instruction and update AR2.
- R) Dummy operation (flush the pipeline).
- E) Load ACC with data read in cycle 7 (ACC = 50h).

Cycle 9:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 8.
- R) Read data from data memory location 64h (20h) which is the location pointed at by AR2 before the update of cycle 8.
- E) Dummy operation (flush the pipeline).

Cycle 10:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 9.
- R) Depends on the instruction fetched in cycle 8.
- E) Add data read in cycle 9 (20h) to data in ACC (50h) and store result in ACC (ACC = 70h).

Example 7–8. Pipeline Operation with ARx Load and NOP Instructions

```
LAR    AR2,#67h
LACC #64h
SAMM   AR2
NOP
NOP
LACC   *–
ADD    *–
.
.
```

Table 7–8. Pipeline Operation with ARx Load and NOP Instructions

Cycle	PC	Pipeline operation				ARP	AR2	ACC
		F	D	R	E			
1	[LACC]	LAR				2	XX	XX
2	[#64h]	LACC	LAR			2	XX	XX
3	[SAMM]	64h	LACC	LAR		2	XX	XX
4	[NOP]	SAMM	dummy	LACC	LAR	2	67h	XX
5	[NOP]	NOP	SAMM	dummy	LACC	2	67h	64h
6	[LACC]	NOP	dummy	SAMM	dummy	2	67h	64h
7	[ADD]	LACC	dummy	dummy	SAMM	2	64h	64h
8		ADD	LACC	dummy	dummy	2	63h	64h
9			ADD	LACC	dummy	2	62h	64h
10				ADD	LACC	2	62h	20h
11					ADD	2	62h	30h

Assume memory location 63h = 10h, 64h = 20h, 65h = 30h, 66h = 40h, and 67h = 50h. The following is the condition of the pipeline for each cycle.

Cycle 1: F) Fetch the LAR instruction and update PC to next instruction.

Cycle 2: F) Fetch the LACC instruction and update PC.
D) Decode the LAR instruction and no ARAU update.

Cycle 3: F) Fetch the second word 64h and update PC.
D) Decode the LACC instruction and no ARAU update.
R) No operand read for the LAR instruction.

Cycle 4: F) Fetch the SAMM instruction and update PC.
D) Dummy operation (previous fetch (F) phase is an operand).
R) No operand read for the LACC instruction.
E) Load AR2 with 67h.

Cycle 5: F) Fetch the NOP instruction and update PC.
D) Decode the SAMM instruction and no ARAU update.
R) Dummy operation (operand fetch on fetch phase).
E) Load ACC with 64h.

Cycle 6:
- F) Fetch the NOP instruction and update PC.
- D) Dummy operation (flush the pipeline).
- R) No operand read for the SAMM instruction.
- E) Dummy operation (operand fetch on fetch phase).

Cycle 7:
- F) Fetch the LACC instruction and update PC.
- D) Dummy operation (flush the pipeline).
- R) Dummy operation (flush the pipeline).
- E) Store data in ACC to AR2 (AR2 = 64h).

Cycle 8:
- F) Fetch the ADD instruction and update PC.
- D) Decode the LACC instruction and update AR2.
- R) Dummy operation (flush the pipeline).
- E) Dummy operation (flush the pipeline).

Cycle 9:
- F) Fetch the next instruction and update PC.
- D) Decode the ADD instruction and update AR2.
- R) Read data from data memory location 64h (20h) which is the location pointed at by AR2 before the update of cycle 8.
- E) Dummy operation (flush the pipeline).

Cycle 10:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 9.
- R) Read data from data memory location 63h (10h) which is the location pointed at by AR2 before the update of cycle 9.
- E) Load ACC with data read in cycle 9 (ACC = 20h).

Cycle 11:
- F) Fetch the next instruction and update PC.
- D) Decode the instruction fetched in cycle 10.
- R) Depends on the instruction fetched in cycle 9.
- E) Add data read in cycle 10 (10h) to data in ACC (20h) and store result in ACC (ACC = 30h).

7.2.4 Pipeline Operation on External Memory Conflict

Since the ’C5x only has one set of external address and data buses, a bus conflict occurs between instruction fetch (F), operand read (R), and execute (E) write phases if both program and data memory are external. While the bus conflict is occurring, a dummy operation can be inserted to eliminate the bus conflict. Example 7–9 shows pipeline operation with a bus conflict and a dummy operation.

In Example 7–9, assume there is no bus conflict between the LACC instruction and the previous instructions. In the operand read (R) phase of LACC, a bus conflict occurs with the fetch of SACL. Therefore, a dummy fetch operation is inserted. In the next fetch (F) phase, the SACL has a bus conflict with the ADD operand read (R) phase. Therefore, the fetch of SACL is delayed again one cycle. Two dummy instruction fetches are inserted between ADD and SACL due to this delay. A similar situation occurred in the execute (E) phase of SACL. Since external memory writes take 3 cycles, during the execution of SACL any instruction fetch or operand read access on the external bus will be delayed for 3 cycles.

Example 7–9. Pipeline Operation with External Bus Conflicts

```
LACC   *+
ADD    *+
SACL   *+, AR2
NOP
.
.
```

Table 7–9. Pipeline Operation with External Bus Conflicts

		Pipeline operation								
Cycle	PC	F	D	R	E	ARP	AR1	AR2	ACC	[802h]
1	[ADD]	LACC				1	800h	803h	XX	FFh
2	[SACL]	ADD	LACC			1	801h	803h	XX	FFh
3	[SACL]	dummy	ADD	LACC		1	802h	803h	XX	FFh
4	[SACL]	dummy	dummy	ADD	LACC	1	802h	803h	10h	FFh
5	NOP	SACL	dummy	dummy	ADD	1	802h	803h	13h	FFh
6		NOP	SACL	dummy	dummy	2	803h	803h	13h	FFh
7			dummy	SACL	dummy	2	803h	802h	13h	FFh
8				dummy	SACL	2	803h	802h	13h	13h
9					dummy	2	803h	802h	13h	13h

Assume memory location 800h = 10h, 801h = 3h, 802h = FFh, and 803h = 6h. The following is the condition of the pipeline for each cycle.

Cycle 1:
- F) Fetch the LACC instruction and update PC to next instruction.

Cycle 2:
- F) Fetch the ADD instruction and update PC.
- D) Decode the LACC instruction and update AR1.

Cycle 3:
- F) Since the read (R) phase occupies the external bus, insert a dummy operation and no update on PC.
- D) Decode the ADD instruction and update AR1.
- R) External data read for the LACC instruction from data memory location 800h (10h) which is the location pointed at by AR1 before the update of cycle 2.

Cycle 4:
- F) Since the read (R) phase occupies the external bus, insert a dummy operation and no update on PC.
- D) Dummy operation from previous fetch phase.
- R) External data read for the ADD instruction from data memory location 801h (3h) which is the location pointed at by AR1 before the update of cycle 3.
- E) Load ACC with data read in cycle 3 (ACC = 10h).

Cycle 5:
- F) Fetch the SACL instruction and update PC.
- D) Dummy operation from previous fetch phase.
- R) Dummy operation from previous decode phase.
- E) Add data read in cycle 4 (3h) to data in ACC (10h) and store result in ACC (ACC = 13h).

Cycle 6:
- F) Fetch the NOP instruction and update PC.
- D) Decode the SACL instruction, and update ARP and AR1.
- R) Dummy operation from previous decode (D) phase.
- E) Dummy operation from previous read (R) phase.

Cycle 7:
- F) Fetch the next instruction and update PC.
- D) Dummy operation (flush the pipeline).
- R) No operand read for the SACL instruction.
- E) Dummy operation from previous read (R) phase.

Cycle 8:
- F) Since the execute (E) phase occupies the external bus and takes 3 cycles for an external write, insert a dummy operation in the next 3 fetch (F) phases and no update on PC.
- D) Decode instruction fetched in cycle 7.

R) Dummy operation (flush the pipeline).

E) Store data in ACC (13h) to external data memory location 802h which is the location pointed at by AR1 before the update of cycle 6.

Cycle 9: F) Dummy operation and no update on PC.

D) Dummy operation from previous fetch (F) phase.

R) Depends on the instruction fetched in cycle 7.

E) Dummy operation (flush the pipeline).

7.3 Pipeline Latency

Memory-mapped registers are accessed by ’C5x instructions in the decode (D) and operand fetch (F) phases of the pipeline. The pipeline operation previously described requires writes to memory-mapped registers, however, latency occurs while accessing and writing to the registers. Table 7–10 outlines the latency required between an instruction that writes to the register via its memory-mapped address and the access of that register by subsequent instructions. Note that all direct accesses to the registers that do not use memory-mapped addressing (such as all ’C2x-compatible instructions: LAR, LT) are pipeline-protected (stalled) and, therefore, do not cause any latency.

The current AR is affected by the NORM instruction during its execute (E) phase of the pipeline. Similar pipeline management, as described above, works in this case. The *-p* option of the assembler detects an AR update or store (SAR) directly after a NORM instruction and inserts NOP instructions automatically to maintain source-code compatibility with the ’C2x.

Table 7–10. Latencies Required

Register	Description	Words	Affects
ARx	Auxiliary registers 0–7	2	Next word uses previous value; second word update gets over written
ARCR	Auxiliary register compare register	2	Next 2 words use previous value
BMAR	Block move address register	1	Next 1 word uses previous value
CBCR	Circular buffer control register	2	Next 2 words cannot be end of buffer
CBER	Circular buffer end registers 1 and 2	2	Next 2 words cannot be end of buffer
CBSR	Circular buffer start registers 1 and 2	2	Next 2 words use previous value
CWSR	Wait-state control register	1	Next 1 word uses previous modes
GREG	Global memory allocation register	1	Next 1 word uses previous map
INDX	Index register	2	Next 2 words use previous value
IOWSR	I/O port wait-state register	1	Next 1 word uses previous count
PDWSR	Program/data wait-state register	1	Next 1 word uses previous count
PMST	Processor mode status register	2	Next 2 words use previous map
ST1	Configuration control (CNF) bit in ST1	2	Next 2 words use previous map
TREG1	Dynamic shift count	1	Next 1 word uses old shift count
TREG2	Dynamic bit address	1	Next 1 word uses old bit address

The ’C5x core CPU supports reconfiguration of memory segments (internal and external) during the execute (E) phase of the pipeline. Therefore, before an instruction utilizes the new configuration, at least two instruction words should follow the instruction that reconfigures memory.

In the following example, assume the current AR = 0200h and RAMB0 (0) = 1.

```
CLRC   CNF       ;Map RAM B0 to data space.
LACC   #01234h   ;ACC = 00001234.
ADD    *         ;ACC = 00001235.
```

Notice the use of the LACC #01234h to fill the 2-word requirement. Because a long-immediate operand is used, this is a 2-word instruction and, therefore, meets the requirement. This also applies to memory configurations controlled by the PMST.

If the main code is running in the B0 block (CNF = 1) and an interrupt service routine not in B0 changes CNF to 0, a RETE will not restore CNF in time to fetch the next instruction from the B0 block. Therefore, in the interrupt service routine, the CNF bit should be set at least 2 words before the RETE.

Chapter 8

Memory

The total memory address range of the ’C5x devices is 224K 16-bit words. The memory space is divided into four individually-selectable memory segments:

- 64K-word program
- 64K-word local data
- 64K-word input/output (I/O) ports
- 32K-word global data

Their parallel architecture lets the ’C5x devices perform three concurrent memory operations in any given machine cycle: fetching an instruction, reading an operand, and writing an operand.

This chapter discusses ’C5x memory configuration and operation.

Topic **Page**

8.1 Memory Space Overview

The ’C5x design is based on the enhanced Harvard architecture, which has multiple memory spaces that can be accessed on two parallel buses. This makes it possible to access both program and data simultaneously. The two parallel buses are the program bus (PB) and data read bus (DB). Each bus accesses different memory spaces for different aspects of the device operation. The ’C5x memory is organized into four individually selectable spaces: program memory, local data memory, global data memory, and I/O ports. These memory spaces compose an address range of 224K words. Within any of these spaces, RAM, ROM, EPROM, EEPROM, or memory-mapped peripherals can reside either on- or off-chip.

The 64K-word program space contains the instructions to be executed. The 64K-word local data space stores data used by the instructions. The 32K-word global data space can share data with other processors within the system or can serve as additional data space. The 64K-word I/O port space interfaces to external memory-mapped peripherals and can also serve as extra data storage space. Within a given machine cycle, the ALU can execute as many as three concurrent memory operations.

The large on-chip memory of the ’C5x devices enhances system performance and integration. This on-chip memory includes ROM in program space, single-access RAM (SARAM) in program and/or data space, and dual-access RAM (DARAM) in program and/or data space. The amount and types of memory available on each device are listed in Table 1–1.

All ’C5x devices have 1056 words of DARAM configured in three blocks and mapped at the same addresses: block 0 (B0) has 512 words at address 0100h–02FFh in local data memory or FE00h–FFFFh in program space; block 1 (B1) has 512 words at address 0300h–04FFh in local data memory; and block 2 (B2) has 32 words at address 0060h–007Fh in local data memory. The DARAM can be read from and written to in the same machine cycle.

The ’C5x devices have different sizes of SARAM (see Table 1–1) which is divided into 2K-word and 1K-word blocks that are contiguous in program or data memory space. The SARAM requires a full machine cycle to perform a read or a write. However, the CPU can read or write one block while accessing another block during the same machine cycle.

The ’C5x devices have different sizes of ROM in program space (see Table 1–1 on page 1-6). This ROM could be maskable ROM or boot ROM. The boot ROM resides in program space at address 0000h and includes a device test (for internal use) and boot code. The maskable ROM is also located in the lowest block of program memory. The ROM is enabled or disabled by the state

of the MP/$\overline{MC}$ pin control input at reset, or by manipulating the MP/$\overline{MC}$ bit in the processor mode status register (PMST) after reset.

The ’C50 (Figure 8–1) includes 2K words of boot ROM, 9K words program/data SARAM, and 1056 words of DARAM. The boot ROM resides in program space at address range 0000h–07FFh. The 9K words of SARAM can be mapped into program or data space and reside at address range 0800h–2BFFh in either space.

The ’C51 (Figure 8–2) removes the 2K-word boot ROM from program memory space and replaces 8K words of program/data SARAM with an 8K-word block of maskable ROM. The ’C51 also includes 1K word of program/data SARAM and 1056 words of DARAM. The 8K words of ROM reside in program space at address range 0000h–1FFFh. The 1K word of SARAM can be mapped into data space (address range 0800h–0BFFh), program space (address range 2000h–23FFh), or both spaces.

The ’C52 (Figure 8–3) includes 4K words of maskable ROM and 1056 words of DARAM. No program/data SARAM is available on the ’C52. The 4K words of ROM reside in program space at address range 0000h–0FFFh.

The ’C53 and ’C53S (Figure 8–4) include 16K words of maskable ROM, 3K words of program/data SARAM, and 1056 words of DARAM. The 16K words of ROM reside in program space at address range 0000h–3FFFh. The 3K words of SARAM can be mapped into data space (address range 0800–13FFh), program space (address range 4000h–4BFFh), or both spaces.

The ’LC56 and ’LC57 (Figure 8–5) include 32K words of maskable ROM, 6K words of program/data SARAM, and 1056 words of DARAM. The 32K words of ROM reside in program space at address range 0000h–7FFFh. The 6K words of SARAM can be mapped into data space (address range 0800–1FFFh), program space (address range 8000h–97FFh), or both spaces.

The ’C57S (Figure 8–6) includes 2K words of boot ROM, 6K words of program/data SARAM, and 1056 words of DARAM. The boot ROM resides in program space at address range 0000h–07FFh. The 6K words of SARAM can be mapped into data space (address range 0800–1FFFh), program space (address range 8000h–97FFh), or both spaces.

Figure 8–1. ’C50 Memory Map

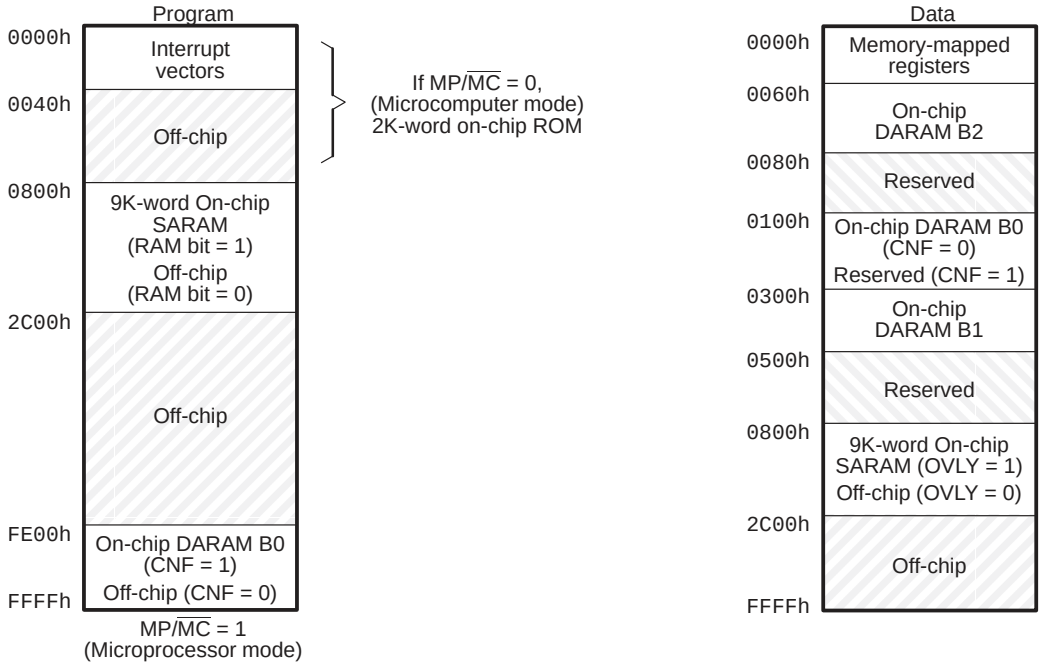


Figure 8–2. ’C51 Memory Map

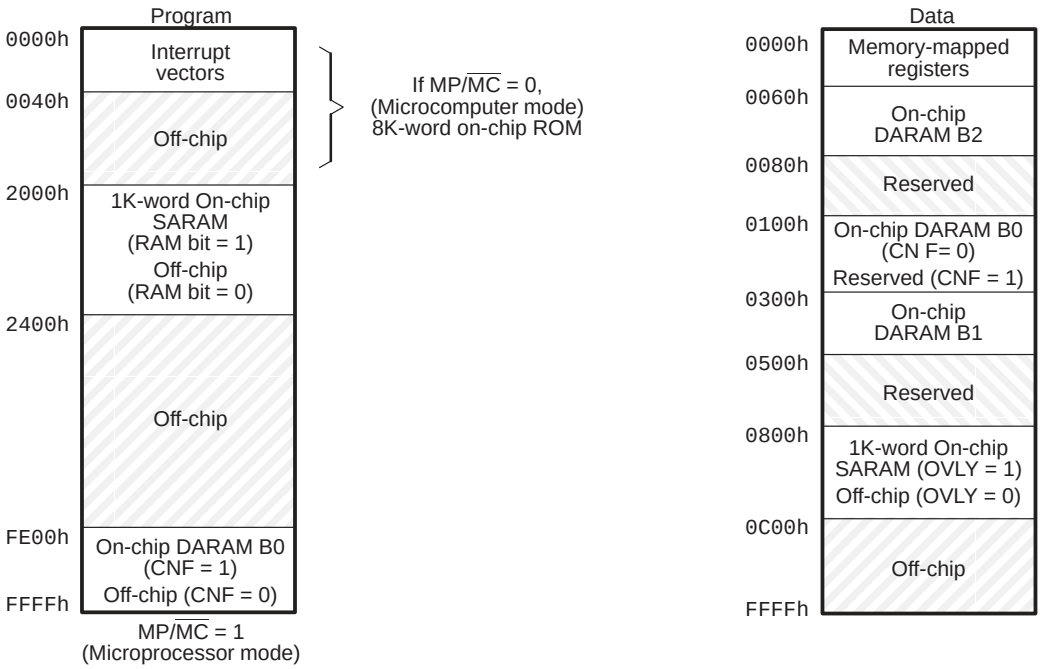


Figure 8–3. ’C52 Memory Map

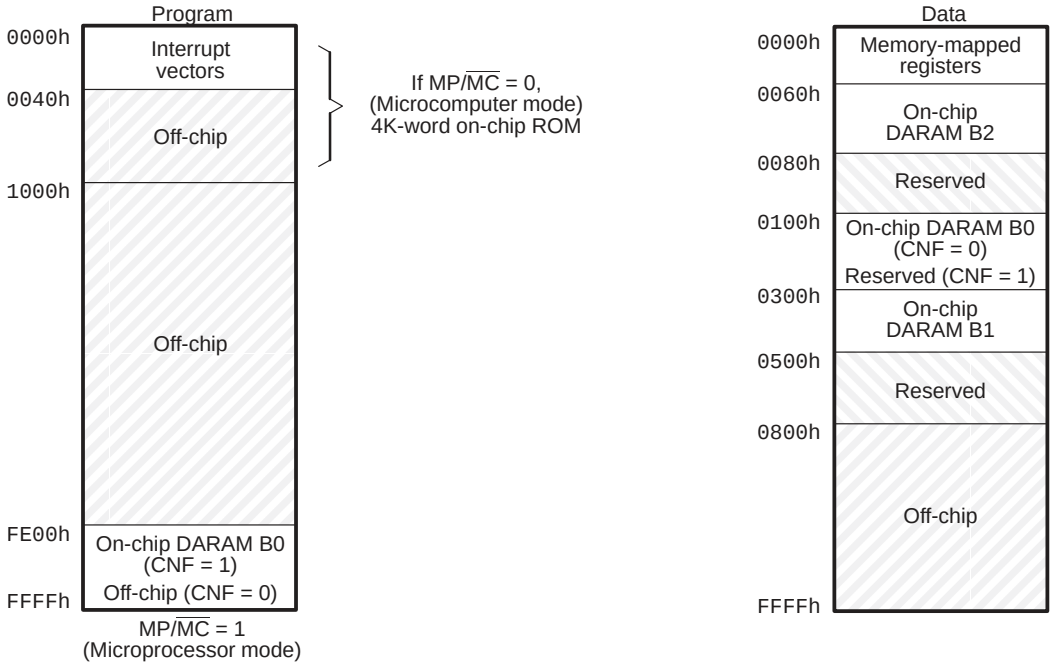


Figure 8–4. ’C53 and ’C53S Memory Map

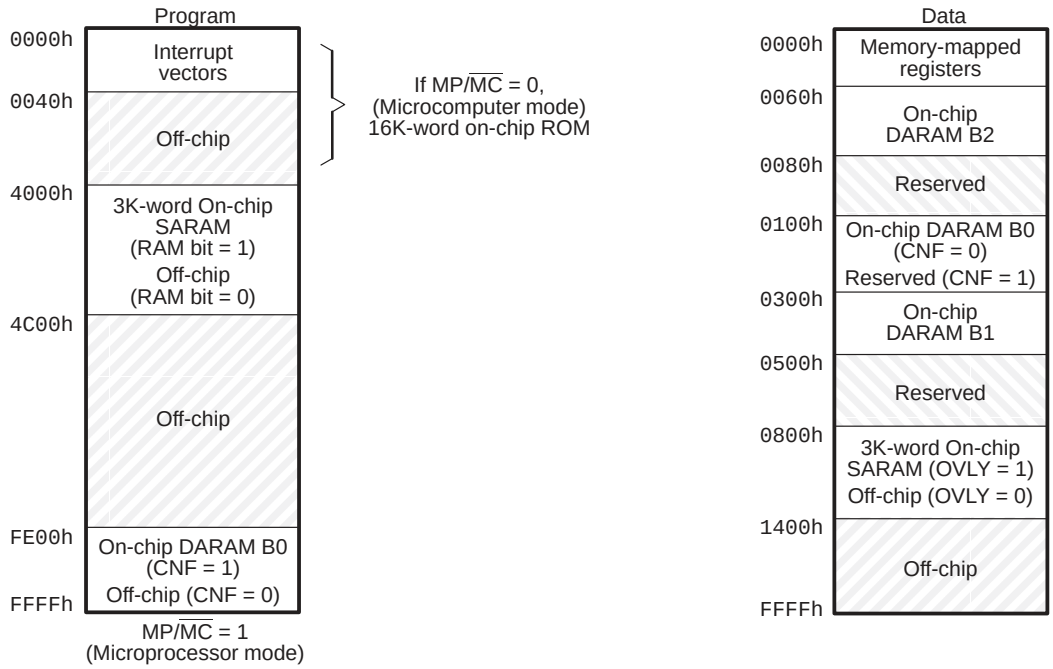


Figure 8–5. ’LC56 and ’LC57 Memory Map

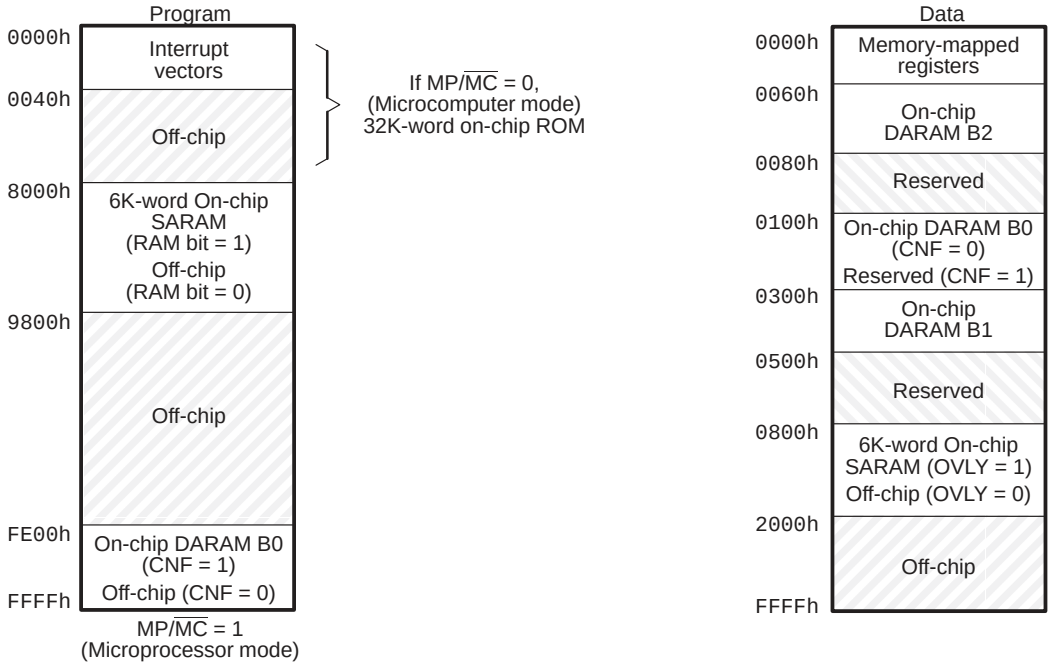
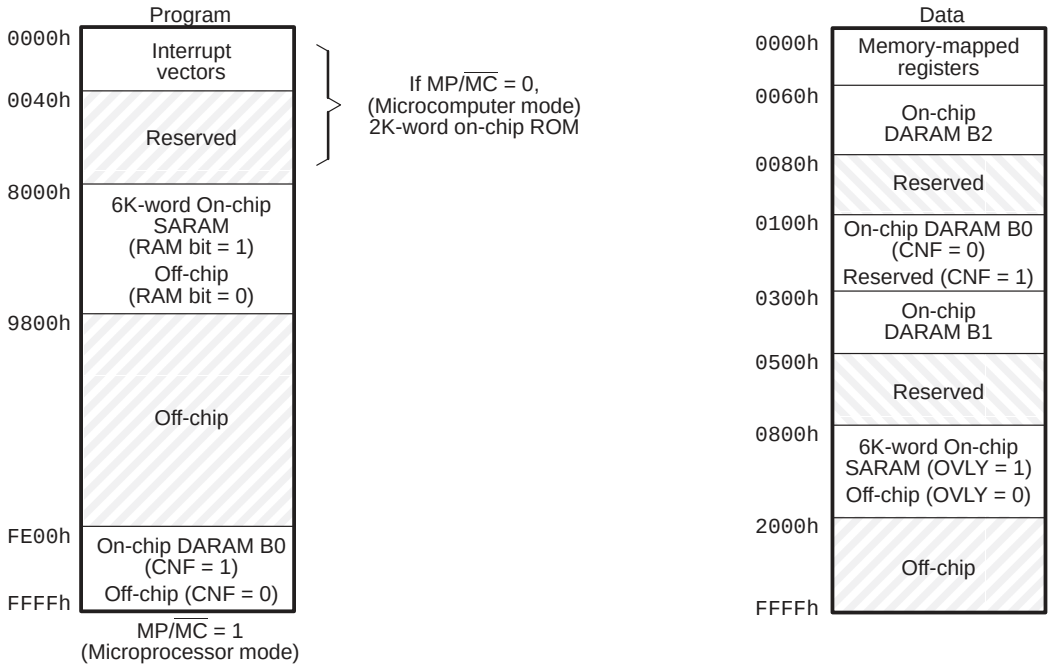


Figure 8–6. ’C57S Memory Map



8.2 Program Memory

The program memory space addresses up to 64K 16-bit words and includes on-chip ROM in program space, SARAM (except in ’C52) in program and/or data space, and DARAM in program and/or data space. The amount and types of memory available on each device are listed in Table 1–1 on page 1-6. Software can configure these memory cells to reside inside (on-chip) or outside (off-chip) of the program address map. When the memory cells are mapped into program space, the ’C5x automatically accesses them when it addresses within their bounds. When the CALU generates an address outside these bounds, the ’C5x automatically generates an external (off-chip) access. These are the advantages of operating from internal (on-chip) memory:

1) Higher performance because no wait states are required for slower external memories.

2) Lower cost than external memory.

3) Lower power than external memory.

The advantage of operating from external (off-chip) memory is the ability to access a larger address space.

8.2.1 Program Memory Configurability

The program memory can reside both on- and off-chip. At reset, the ’C5x device configuration is set by the level on the MP/$\overline{\text{MC}}$ pin. If this pin is high, the device is configured as a microprocessor, and the on-chip ROM is not addressed. If this pin is low, the device is configured as a microcomputer, and the on-chip ROM is enabled.

The ’C5x devices fetch their reset vector in program memory at address location 0000h; so, if the device is operating as a microcomputer, it starts running from on-chip ROM. If the device is operating as a microprocessor, it starts running from off-chip memory. Once the program is running, the device configuration can be changed by setting or clearing the MP/$\overline{\text{MC}}$ bit in the PMST. Note that the MP/$\overline{\text{MC}}$ pin is sampled only at reset. The following instruction removes the ROM from program space by setting the MP/$\overline{\text{MC}}$ bit in the PMST to 1:

```
OPL#8,PMST   ;Remove boot ROM from program space.
```

Code can be submitted to be masked into the on-chip ROM for ’C51, ’C52, ’C53, ’C56, and ’C57 devices. The process-masked ROM cell requires ROM codes to be submitted to Texas Instruments for implementation in the device, as detailed in Appendix F, *Submitting ROM Codes to TI*.

At reset, the SARAM and the 512-word DARAM block B0 are not resident in program space. You make the SARAM resident in program space by setting the RAM bit in the PMST. When the RAM bit is set, the RAM cells become addressable in program space. You make the DARAM block B0 resident in program space (address range FE00h–FFFFh) by setting the CNF bit in the ST1. The following instructions map the SARAM and DARAM blocks into program space by setting the appropriate bit in the registers:

```
OPL   #010h,PMST   ;Map 'C5x single-access memory
                   ;in program space.
SETC  CNF          ;Map B0 to program space.
```

Table 8–1 through Table 8–6 show program memory configurations available on the 'C5x devices. Note that all addresses are specified in hexadecimal.

Table 8–1. 'C50 Program Memory Configuration

Bit values			ROM (2K-words)	SARAM (9K-words)	DARAM B0 (512-words)	Off-Chip
CNF	RAM	MP/$\overline{MC}$				
0	0	0	0000–07FF	Off-chip	Off-chip	0800–FFFF
0	0	1	Off-chip	Off-chip	Off-chip	0000–FFFF
0	1	0	0000–07FF	0800–2BFF	Off-chip	2C00–FFFF
0	1	1	Off-chip	0800–2BFF	Off-chip	0000–07FF, 2C00–FFFF
1	0	0	0000–07FF	Off-chip	FE00–FFFF	0800–FDFF
1	0	1	Off-chip	Off-chip	FE00–FFFF	0000–FDFF
1	1	0	0000–07FF	0800–2BFF	FE00–FFFF	2C00–FDFF
1	1	1	Off-chip	0800–2BFF	FE00–FFFF	0000–07FF, 2C00–FDFF

Table 8–2. ’C51 Program Memory Configuration

Bit values			ROM (8K-words)	SARAM (1K-words)	DARAM B0 (512-words)	Off-Chip
CNF	RAM	MP/$\overline{MC}$				
0	0	0	0000–1FFF	Off-chip	Off-chip	2000–FFFF
0	0	1	Off-chip	Off-chip	Off-chip	0000–FFFF
0	1	0	0000–1FFF	2000–23FF	Off-chip	2400–FFFF
0	1	1	Off-chip	2000–23FF	Off-chip	0000–1FFF, 2400–FFFF
1	0	0	0000–1FFF	Off-chip	FE00–FFFF	2000–FDFF
1	0	1	Off-chip	Off-chip	FE00–FFFF	0000–FDFF
1	1	0	0000–1FFF	2000–23FF	FE00–FFFF	2400–FDFF
1	1	1	Off-chip	2000–23FF	FE00–FFFF	0000–1FFF, 2400–FDFF

Table 8–3. ’C52 Program Memory Configuration

Bit values			ROM (4K-words)	SARAM	DARAM B0 (512-words)	Off-Chip
CNF	RAM	MP/$\overline{MC}$				
0	X	0	0000–0FFF	None	Off-chip	1000–FFFF
0	X	1	Off-chip	None	Off-chip	0000–FFFF
1	X	0	0000–0FFF	None	FE00–FFFF	1000–FDFF
1	X	1	Off-chip	None	FE00–FFFF	0000–FDFF

Legend: X = Don’t care condition

Table 8–4. ’C53 and ’C53S Program Memory Configuration

Bit values			ROM (16K-words)	SARAM (3K-words)	DARAM B0 (512-words)	Off-Chip
CNF	RAM	MP/$\overline{MC}$				
0	0	0	0000–3FFF	Off-chip	Off-chip	4000–FFFF
0	0	1	Off-chip	Off-chip	Off-chip	0000–FFFF
0	1	0	0000–3FFF	4000–4BFF	Off-chip	4C00–FFFF
0	1	1	Off-chip	4000–4BFF	Off-chip	0000–3FFF, 4C00–FFFF
1	0	0	0000–3FFF	Off-chip	FE00–FFFF	4000–FDFF
1	0	1	Off-chip	Off-chip	FE00–FFFF	0000–FDFF
1	1	0	0000–3FFF	4000–4BFF	FE00–FFFF	4C00–FDFF
1	1	1	Off-chip	4000–4BFF	FE00–FFFF	0000–3FFF, 4C00–FDFF

Table 8–5. ’LC56 and ’LC57 Program Memory Configuration

Bit values			ROM (32K-words)	SARAM (6K-words)	DARAM B0 (512-words)	Off-Chip
CNF	RAM	MP/$\overline{MC}$				
0	0	0	0000–7FFF	Off-chip	Off-chip	8000–FFFF
0	0	1	Off-chip	Off-chip	Off-chip	0000–FFFF
0	1	0	0000–7FFF	8000–97FF	Off-chip	9800–FFFF
0	1	1	Off-chip	8000–97FF	Off-chip	0000–7FFF, 9800–FFFF
1	0	0	0000–7FFF	Off-chip	FE00–FFFF	8000–FDFF
1	0	1	Off-chip	Off-chip	FE00–FFFF	0000–FDFF
1	1	0	0000–7FFF	8000–97FF	FE00–FFFF	9800–FDFF
1	1	1	Off-chip	8000–97FF	FE00–FFFF	0000–7FFF, 9800–FDFF

Table 8–6. ’C57S Program Memory Configuration

Bit values			ROM	SARAM	DARAM B0	
CNF	RAM	MP/$\overline{MC}$	(2K-words)	(6K-words)	(512-words)	Off-Chip
0	0	0	0000–07FF	Off-chip	Off-chip	8000–FFFF
0	0	1	Off-chip	Off-chip	Off-chip	0000–FFFF
0	1	0	0000–07FF	8000–97FF	Off-chip	9800–FFFF
0	1	1	Off-chip	8000–97FF	Off-chip	0000–7FFF, 9800–FFFF
1	0	0	0000–07FF	Off-chip	FE00–FFFF	8000–FDFF
1	0	1	Off-chip	Off-chip	FE00–FFFF	0000–FDFF
1	1	0	0000–07FF	8000–97FF	FE00–FFFF	9800–FDFF
1	1	1	Off-chip	8000–97FF	FE00–FFFF	0000–7FFF, 9800–FDFF

8.2.2 Program Memory Address Map

The interrupt vectors are addressed in program space. These vectors are soft — meaning that the processor, when taking the trap, loads the program counter (PC) with the trap address and executes code at the vector location. Two words are reserved at each vector location for a branch instruction to the appropriate interrupt service routine (ISR). Table 8–7 lists the interrupt vector addresses after reset.

At reset, the interrupt vector is mapped absolutely to address 0000h in program space. However, the interrupt vector can be remapped to the beginning of any 2K-word page in program space after reset. To do this, load the interrupt vector pointer (IPTR) bits in the PMST with the appropriate 2K-word page boundary address. After IPTR is loaded, any user interrupt vector is mapped to the new 2K-word page. For example:

```
OPL#05800h,PMST    ;Remap vectors to start at 5800h.
```

In this example, the interrupt vectors move to off-chip program space beginning at address 5800h. Any subsequent interrupt (except for a reset) will fetch its interrupt vector from that new location. For example, if, after loading the IPTR, an $\overline{INT2}$ occurs, the interrupt service routine vector will be fetched from address 5804h in program space as opposed to address 0004h. This feature lets you move the desired vectors out of the boot ROM and then remove the ROM from the memory map. Once the system code is booted into the system from the boot-loader code resident in ROM, the application reloads the IPTR

with a value pointing to the new vectors. In the above example, the OPL instruction is used to modify the IPTR bits in the PMST. This example assumes that the IPTR bits are currently cleared. To assure that the correct value for IPTR is set, the bits must be cleared before this instruction is executed.

Note:

The reset vector can not be remapped, because reset loads the IPTR with 0. Therefore, the reset vector will always be fetched at location 0000h in program memory. In addition, for the ’C51 and ’C53, 100 words are reserved in the on-chip ROM for device-testing purposes. Application code written to be implemented in on-chip ROM must reserve these 100 words at the top of the ROM addresses.

Table 8–7. ’C5x Interrupt Vector Addresses

	Location			
Name	**Dec**	**Hex**	**Priority**	**Function**
$\overline{\text{RS}}$	0	0	1 (highest)	External nonmaskable reset signal
$\overline{\text{INT1}}$	2	2	3	External user interrupt #1
$\overline{\text{INT2}}$	4	4	4	External user interrupt #2
$\overline{\text{INT3}}$	6	6	5	External user interrupt #3
TINT	8	8	6	Internal timer interrupt
RINT	10	A	7	Serial port receive interrupt
XINT	12	C	8	Serial port transmit interrupt
TRNT†	14	E	9	TDM port receive interrupt
TXNT‡	16	10	10	TDM port transmit interrupt
$\overline{\text{INT4}}$	18	12	11	External user interrupt #4
——	20–23	14–17	N/A	Reserved
HINT	24	18	——	HINT (’C57 only)
——	26–33	1A–21	N/A	Reserved
TRAP	34	22	N/A	Software trap instruction
$\overline{\text{NMI}}$	36	24	2	Nonmaskable interrupt
——	38–39	26–27	N/A	Reserved for emulation and test
——	40–63	28–3F	N/A	Software interrupts

† RINT2 on ’C52; BRNT on ’C56/C57
‡ XINT2 on ’C52; BXNT on ’C56/C57

8.2.3 Program Memory Addressing

The program memory space contains the code for applications. It can also hold table information and immediate operands. The program memory is accessed only by the PAB. The address for this bus is generated by the PC when instructions and long immediate operands are accessed. The PAB can also be loaded with long immediate, low accumulator, or registered addresses for block transfers, multiply/accumulates, and table read/writes.

The ’C5x fetches instructions by putting the PC on the PAB and reading the appropriate location in program memory. While the read is executing, the PC is incremented for the next fetch. If there is a program address discontinuity (for example, branch, call, return, interrupt, or block repeat), the appropriate address is loaded into the PC. The PC is also loaded when operands are fetched from program memory, which occurs when the device reads from (TBLR) or writes to (TBLW) tables, when it transfers data to (BLPD) or from (BLDP) data space, or when it uses the program bus to fetch a second multiplicand (MAC, MACD, MADS, and MADD). See Section 4.1, *Program Counter (PC)*, on page 4-2.

The data used as instruction operands is obtained in one of the following addressing modes:

- ❏ The direct addressing mode
- ❏ The indirect addressing mode
- ❏ The short immediate addressing mode
- ❏ The long immediate addressing mode
- ❏ The dedicated-register addressing mode
- ❏ The memory-mapped register addressing mode

Refer to Chapter 5, *Addressing Modes*, for a discussion about the addressing modes.

Address Visibility

The address visibility (AVIS) feature can trace the address flow of a program externally and can be used for debugging during program development. It is enabled after reset and can be disabled by setting the AVIS bit in the PMST and enable it by clearing the AVIS bit. The address visibility mode sends the program address out to the address pins of the device, even when on-chip program memory is addressed. Note that the memory control signals ($\overline{PS}$, $\overline{RD}$, etc.) are not active in this mode.

Instruction addresses can be externally clocked with the falling edge of the instruction acquisition ($\overline{\text{IAQ}}$) pin (refer to the TMS320C5x data sheet for $\overline{\text{IAQ}}$ timings). These instruction addresses include both words of a 2-word instruction but do not include block transfers, table reads, or multiply/accumulate operands. The address visibility mode also allows a specific interrupt trap to be decoded in conjunction with the interrupt acknowledge ($\overline{\text{IACK}}$) pin. While $\overline{\text{IACK}}$ is low, address pins A1–A4 can be decoded to identify which interrupt is being acknowledged (refer to the TMS320C5x data sheet for $\overline{\text{IACK}}$ timings).

Once the system is debugged, you can disable the address visibility mode by setting the AVIS bit. Disabling the address visibility mode lowers the power consumption of the device and the RF noise of the system. Note that if the processor is running while $\overline{\text{HOLDA}}$ is active (HM = 0), the program address is not available at the address pins, regardless of the address visibility mode.

8.2.4 Program Memory Protection Feature

The program memory protection feature prevents an instruction fetched from off-chip memory from reading or writing on-chip program memory. The pipeline controller tracks instructions fetched from off-chip memory, and, if the operand address resides in on-chip program space, the instruction reads invalid data off the bus. The protection feature implements these limitations:

- ❑ Instructions fetched from off-chip memory cannot read or write on-chip single-access and read-only program memory.
- ❑ Instructions fetched from DARAM block B0 cannot read or write on-chip single-access and read-only program memory.
- ❑ Coefficients for off-chip multiply/accumulate instructions cannot reside in on-chip single-access and read-only program memory.
- ❑ The on-chip single-access memory cannot be mapped to program space.
- ❑ The on-chip single-access memory cannot be mapped to data space.
- ❑ External DMA cannot be used.
- ❑ The emulator cannot access on-chip program memory.
- ❑ The program memory address range that corresponds to the on-chip single-access RAM is not available for external memory.

This feature can be used with the on-chip ROM to secure program code that is stored in external (off-chip) memory. The ROM code can include a decryption algorithm that takes encrypted off-chip code, decrypts it, and stores the routine in on-chip single-access program RAM. This process-mask option, like the ROM, must be submitted to Texas Instruments for implementation.

8.3 Local Data Memory

The local data memory space on the ’C5x addresses up to 64K 16-bit words. All the ’C5x devices have 1056 words of DARAM but different sizes of SARAM. The amount and types of memory available on each device are listed in Table 1–1 on page 1-6. You can use software to configure these memory cells to reside inside (on-chip) or outside (off-chip) of the local data address map. When the memory cells are mapped into local data space, the ’C5x automatically accesses them when it addresses within their bounds. When the CALU generates an address outside these bounds, the ’C5x automatically generates an external (off-chip) access. These are the advantages of operating from internal (on-chip) memory:

1) Higher performance because no wait states are required for slower external memories.
2) Higher performance because of more efficient pipeline operation.
3) Lower cost than external memory.
4) Lower power than external memory.

The advantage of operating from external (off-chip) memory is the ability to access a larger address space.

8.3.1 Local Data Memory Configurability

The local data memory can reside both on- and off-chip. At reset, the ’C5x device configuration maps the 1056 words of DARAM into local data space. DARAM block B0 can be reconfigured into program space by setting the CNF bit in ST1. SARAM can be mapped into data space by setting the OVLY bit in the PMST.

Table 8–8 through Table 8–12 show local data memory configurations available on the ’C5x devices. Note that all locations in the address range 0000h–0800h that are not mapped into on-chip memory are on-chip reserved locations. Address range 0000h–004Fh contains on-chip memory-mapped registers, and address range 0050h–005Fh contains the memory-mapped I/O ports. Note that all addresses are specified in hexadecimal.

Table 8–8. ’C50 Local Data Memory Configuration

Bit values		Registers	DARAM B2	DARAM B0	DARAM B1	SARAM	
CNF	OVLY	(96-words)	(32-words)	(512-words)	(512-words)	(9K-words)	Off-Chip
0	0	0000–005F	0060–007F	0100–02FF	0300–04FF	Off-chip	0800–FFFF
0	1	0000–005F	0060–007F	0100–02FF	0300–04FF	0800–2BFF	2C00–FFFF
1	0	0000–005F	0060–007F	Reserved	0300–04FF	Off-chip	0800–FFFF
1	1	0000–005F	0060–007F	Reserved	0300–04FF	0800–2BFF	2C00–FFFF

Table 8–9. ’C51 Local Data Memory Configuration

Bit values		Registers	DARAM B2	DARAM B0	DARAM B1	SARAM	
CNF	OVLY	(96-words)	(32-words)	(512-words)	(512-words)	(1K-words)	Off-Chip
0	0	0000–005F	0060–007F	0100–02FF	0300–04FF	Off-chip	0800–FFFF
0	1	0000–005F	0060–007F	0100–02FF	0300–04FF	0800–0BFF	0C00–FFFF
1	0	0000–005F	0060–007F	Reserved	0300–04FF	Off-chip	0800–FFFF
1	1	0000–005F	0060–007F	Reserved	0300–04FF	0800–0BFF	0C00–FFFF

Table 8–10. ’C52 Local Data Memory Configuration

Bit values		Registers	DARAM B2	DARAM B0	DARAM B1		
CNF	OVLY	(96-words)	(32-words)	(512-words)	(512-words)	SARAM	Off-Chip
0	X	0000–005F	0060–007F	0100–02FF	0300–04FF	None	0800–FFFF
1	X	0000–005F	0060–007F	Reserved	0300–04FF	None	0800–FFFF

Legend: X = Don’t care condition

Table 8–11. ’C53 and ’C53S Local Data Memory Configuration

Bit values		Registers	DARAM B2	DARAM B0	DARAM B1	SARAM	
CNF	OVLY	(96-words)	(32-words)	(512-words)	(512-words)	(3K-words)	Off-Chip
0	0	0000–005F	0060–007F	0100–02FF	0300–04FF	Off-chip	0800–FFFF
0	1	0000–005F	0060–007F	0100–02FF	0300–04FF	0800–13FF	1400–FFFF
1	0	0000–005F	0060–007F	Reserved	0300–04FF	Off-chip	0800–FFFF
1	1	0000–005F	0060–007F	Reserved	0300–04FF	0800–13FF	1400–FFFF

Table 8–12. 'LC56, 'LC57, and 'C57S Local Data Memory Configuration

Bit values		Registers	DARAM B2	DARAM B0	DARAM B1	SARAM	
CNF	OVLY	(96-words)	(32-words)	(512-words)	(512-words)	(6K-words)	Off-Chip
0	0	0000–005F	0060–007F	0100–02FF	0300–04FF	Off-chip	0800–FFFF
0	1	0000–005F	0060–007F	0100–02FF	0300–04FF	0800–1FFF	2000–FFFF
1	0	0000–005F	0060–007F	Reserved	0300–04FF	Off-chip	0800–FFFF
1	1	0000–005F	0060–007F	Reserved	0300–04FF	0800–1FFF	2000–FFFF

8.3.2 Local Data Memory Address Map

The 64K words of local data memory space include the memory-mapped registers for the device. The memory-mapped registers reside in data page 0. Data page 0 has five sections of register banks: CPU registers, peripheral registers, test/emulation reserved area, I/O space, and scratch-pad RAM. Table 8–13 lists the addresses of data page 0.

- The 28 CPU registers can be accessed with zero wait states. Some of these registers can be accessed through paths other than the data bus — for example, auxiliary registers can be loaded by the auxiliary register arithmetic unit (ARAU) by using the LAR instruction.
- The peripheral registers are the control and data registers used in the peripheral circuits. These registers reside on a dedicated peripheral bus structure called the TI Bus. They require one wait state when accessed.
- The test/emulation reserved area is used by the test and emulation systems for special information transfers.

Writing to the test/emulation reserved area can cause the device to change its operational mode and, therefore, affect the operation of the application.

- The I/O space provides access to 16 words of I/O space (other than IN and OUT instructions) via the more extensive addressing modes available within the data space. For example, the SAMM instruction can write to an I/O memory-mapped port as an OUT instruction does. The external interface functions as if an OUT instruction occurred ($\overline{IS}$ active). Port addresses reside off-chip and are subject to external wait states. They are also affected by the on-chip software wait-state generator, like any other nonmemory-mapped I/O port.

- The 32-word scratch-pad RAM of DARAM block B2 can be used to hold overhead variables so that the larger blocks of RAM are not fragmented. This RAM block supports dual-access operations and can be addressed via the memory-mapped addressing mode or any data memory addressing mode.

Table 8–13. Data Page 0 Address Map — CPU Registers

Address			
Dec	**Hex**	**Name**	**Description**
0–3	0–3	——	Reserved
4	4	IMR	Interrupt mask register
5	5	GREG	Global memory allocation register
6	6	IFR	Interrupt flag register
7	7	PMST	Processor mode status register
8	8	RPTC	Repeat counter register
9	9	BRCR	Block repeat counter register
10	A	PASR	Block repeat program address start register
11	B	PAER	Block repeat program address end register
12	C	TREG0	Temporary register 0 (used for multiplicand)
13	D	TREG1	Temporary register 1 (used for dynamic shift count)
14	E	TREG2	Temporary register 2 (used as bit pointer in dynamic bit test)
15	F	DBMR	Dynamic bit manipulation register
16	10	AR0	Auxiliary register 0
17	11	AR1	Auxiliary register 1
18	12	AR2	Auxiliary register 2
19	13	AR3	Auxiliary register 3
20	14	AR4	Auxiliary register 4
21	15	AR5	Auxiliary register 5
22	16	AR6	Auxiliary register 6
23	17	AR7	Auxiliary register 7
24	18	INDX	Index register
25	19	ARCR	Auxiliary register compare register
26	1A	CBSR1	Circular buffer 1 start register

Table 8–13. Data Page 0 Address Map — CPU Registers (Continued)

Address			
Dec	**Hex**	**Name**	**Description**
27	1B	CBER1	Circular buffer 1 end register
28	1C	CBSR2	Circular buffer 2 start register
29	1D	CBER2	Circular buffer 2 end register
30	1E	CBCR	Circular buffer control register
31	1F	BMAR	Block move address register
32–35	20–23	——	Memory-mapped serial port registers†
36–42	24–2A	——	Memory-mapped peripheral registers†
43–47	2B–2F	——	Reserved for test/emulation
48–55	30–37	——	Memory-mapped serial port registers†
56–79	38–4F	——	Reserved
80–95	50–5F	——	Memory-mapped I/O ports†
96–127	60–7F	——	Scratch-pad RAM (DARAM block B2)

† See subsection 9.1.1, *Memory-Mapped Peripheral Registers and I/O Ports*, on page 9-2

8.3.3 Local Data Memory Addressing

The local data space address generation is controlled by the decode of the current instruction. Local data memory is read via data address bus 1 (DAB) on instructions with only one data memory operand and via program address bus (PAB) on instructions with a second data memory operand. An instruction operand is provided to the CALU as described in subsection 8.2.3 on page 8-13. However, data memory addresses are generated in one of the following ways:

- The direct addressing mode
- The indirect addressing mode
- The long immediate operand addressing mode
- The dedicated-register addressing mode
- The memory-mapped register addressing mode

Refer to Chapter 5, *Addressing Modes*, for a discussion about the addressing modes.

8.4 Global Data Memory

For multiprocessing applications, the ’C5x devices can allocate global data memory space and communicate with that space via the $\overline{BR}$ (bus request) and READY control signals. This capability can be used to extend the data memory address map by overlaying the address space.

Since global memory is shared by more than one processor, access to it must be arbitrated. When global memory is used, the processor’s address space is divided into local and global sections. The local section is used by the processor to perform its individual function, and the global section is used to communicate with other processors. This implementation facilitates shared data multiprocessing in which data is transferred between two or more processors. Unlike a direct memory access (DMA) between two processors, reading or writing global memory does not require that one of the processors be halted.

8.4.1 Global Data Memory Configurability

The global memory allocation register (GREG) specifies part of the ’C5x data memory as global external memory. The 8-bit GREG is memory-mapped to data memory address location 05h and is connected to the eight LSBs of the internal data bus. The upper eight bits of location 05h are unused and are read as 1s.

The contents of GREG determine the size (between 256 and 32K words) of the global memory space. The legal values of GREG and the corresponding local and global memory spaces are listed in Table 8–14.

Note:

In Table 8–14 all addresses are specified in hexadecimal; values in GREG other than those listed will lead to fragmented memory maps and should be avoided.

8.4.2 Global Data Memory Addressing

When a data memory address, either direct or indirect, corresponds to a global data memory address (as defined by GREG), $\overline{BR}$ is asserted low with $\overline{DS}$ to indicate that the ’C5x device is starting a global memory access. External logic then arbitrates for control of the global memory, asserting READY when the ’C5x device has control. The length of the memory cycle is controlled by the READY signal. In addition, the software wait-state generators can be used to extend the access times for slower, external memories. The wait-state generators corresponding to the overlapped memory address space in local data space generate the wait states for the corresponding addresses in global data memory space.

Table 8–14. Global Data Memory Configurations

	Local Memory		Global Memory	
GREG value	**Range**	**# Words**	**Range**	**# Words**
0000 00XX	0000–FFFF	65 536	—	0
1000 0000	0000–7FFF	32 768	8000–FFFF	32 768
1100 0000	0000–BFFF	49 152	C000–FFFF	16 384
1110 0000	0000–DFFF	57 344	E000–FFFF	8192
1111 0000	0000–EFFF	61 440	F000–FFFF	4096
1111 1000	0000–F7FF	63 488	F800–FFFF	2048
1111 1100	0000–FBFF	64 512	FC00–FFFF	1024
1111 1110	0000–FDFF	65 024	FE00–FFFF	512
1111 1111	0000–FEFF	65 280	FF00–FFFF	256

Legend: X = Don't care condition

8.5 Input/Output (I/O) Space

The ’C5x devices support an I/O address space of 64K 16-bit parallel input and output (I/O) ports. The I/O ports allow access to peripherals typically used in DSP applications such as codecs, digital-to-analog (D/A) converters, and analog-to-digital (A/D) converters. This section discusses addressing I/O ports and interfacing I/O ports to external devices.

8.5.1 Addressing I/O Ports

Access to external parallel I/O ports is multiplexed over the same address and data bus for program/data memory accesses. All 64K I/O ports can be accessed via the IN and OUT instructions, as shown in the following example:

```
IN DAT7,0FFFEh  ;Read data to data memory from external
                ;device on port 65534.
OUTDAT7,0FFFFh  ;Write data from data memory to external
                ;device on port 65535.
```

Sixteen of the 64K I/O ports are memory-mapped to data memory address locations 50h–5Fh. The I/O ports can be accessed by using the IN and OUT instructions or any instruction that reads or writes a location in data memory space. See Section 9.6, *Parallel I/O Ports*, on page 9-22.

The access times to I/O ports can be modified through the software wait-state registers (IOWSR and CWSR). The BIG bit in the CWSR determines how the I/O space is partitioned. See Section 9.4, *Software-Programmable Wait-State Generators*, on page 9-13.

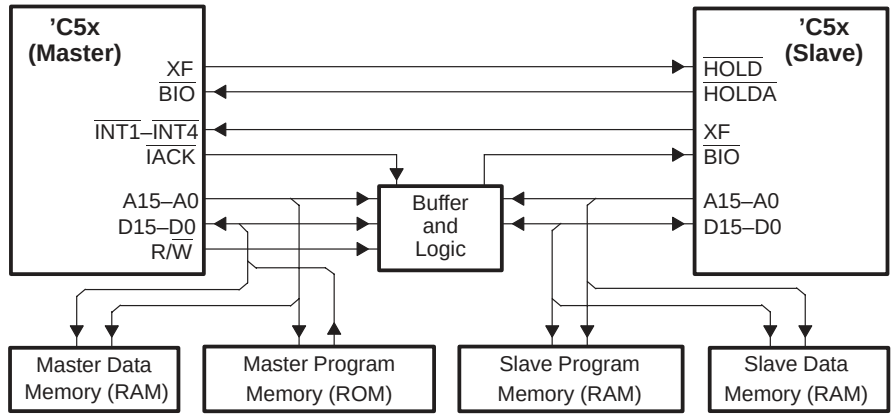
8.6 Direct Memory Access (DMA)

The ’C5x supports multiprocessing designs that require direct memory access (DMA) of external memory or of on-chip single-access RAM. The DMA feature can be used for multiprocessing by temporarily halting the execution of one or more processors to allow another processor to read from or write to local off-chip memory or on-chip single-access RAM. External memory access can be controlled via the $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ signals and on-chip RAM access via the $\overline{\text{HOLD}}$, $\overline{\text{HOLDA}}$, R/$\overline{\text{W}}$, $\overline{\text{STRB}}$, $\overline{\text{BR}}$, and $\overline{\text{IAQ}}$ signals.

8.6.1 DMA in a Master-Slave Configuration

Multiprocessing systems typically utilize a master-slave configuration. The master may initialize a slave by downloading a program into the slave’s program memory space and/or may provide the slave with the necessary data by using external memory to complete a task. In a typical ’C5x DMA scheme, the master may be a general-purpose CPU, another ’C5x, or even an A/D converter. A simple ’C5x master-slave configuration is shown in Figure 8–7.

Figure 8–7. Direct Memory Access Using a Master-Slave Configuration



The ’C5x master device takes complete control of the slave’s external memory by asserting the slave’s $\overline{\text{HOLD}}$ low via the master’s external flag (XF) pin. This causes the slave to place its address, data, and control lines in a high-impedance state.

When the master gains control of the slave’s buses, the slave asserts $\overline{\text{HOLDA}}$. This signal may be tied to the master $\overline{\text{BIO}}$ pin. The slave’s XF pin can indicate to the master when the slave has finished performing its task and needs to be reprogrammed or requires additional data to continue processing. In a multiple-slave configuration, priority of each slave’s task can be determined by

connecting the slave's XF signals to the appropriate $\overline{\text{INT1}}$–$\overline{\text{INT4}}$ pin on the master device. The external bus interface of the slave device is put in high-impedance mode when its $\overline{\text{HOLDA}}$ signal is asserted. Once $\overline{\text{HOLDA}}$ goes active, the $\overline{\text{IAQ}}$ pin does not indicate an instruction acquisition. While the $\overline{\text{HOLDA}}$ is active and the CPU is in hold mode (HM = 0), the CPU continues running code from internal memory (internal ROM or single/dual access RAM). If the CPU is not in hold mode (HM = 1), the CPU halts internal execution. See Section 4.9, *Reset*, on page 4-45 for interaction between $\overline{\text{HOLD}}$, $\overline{\text{RS}}$, and external interrupts.

8.6.2 External DMA

The 'C5x also provides access of the on-chip single-access RAM by external devices through a mechanism called external DMA. External DMA requires the following signals:

A(15–0)	Address inputs when $\overline{\text{HOLDA}}$ and $\overline{\text{BR}}$ are low.
$\overline{\text{BR}}$	Bus request signal externally driven low in hold mode to indicate a request for access.
D(15–0)	DMA data.
$\overline{\text{HOLD}}$	External request for control of address, data, and control lines.
$\overline{\text{HOLDA}}$	Indication to external circuitry that the memory address, data, and control lines are in high impedance, allowing external access.
$\overline{\text{IAQ}}$	Acknowledge $\overline{\text{BR}}$ request for access while $\overline{\text{HOLDA}}$ is low.
R/$\overline{\text{W}}$	Read/write signal indicates the data bus direction for DMA reads (high) and DMA writes (low).
$\overline{\text{STRB}}$	When $\overline{\text{IAQ}}$ and $\overline{\text{HOLDA}}$ are low, $\overline{\text{STRB}}$ selects the memory access and determines its duration.

To access the 'C5x on-chip SARAM, a master processor must control the 'C5x device. The master processor initiates a DMA transfer by asserting the 'C5x device $\overline{\text{HOLD}}$ low. The 'C5x responds by asserting $\overline{\text{HOLDA}}$. The master gains control of the 'C5x bus and access to the SARAM by asserting $\overline{\text{BR}}$ low. The 'C5x responds by asserting $\overline{\text{IAQ}}$ low to acknowledge the access. Once access is granted, the master drives the R/$\overline{\text{W}}$ signal to indicate the direction of the transfer. On a DMA write, the master must drive the address and data lines for a write. On a DMA read, the master must drive the address lines and latch the data. Each memory access (read or write) is selected when $\overline{\text{STRB}}$ is low. External access wait states are added by extending the $\overline{\text{STRB}}$ signal. The address decode of the DMA access includes only address lines A13–A0 (A14 and A15 are ignored). Table 8–15 lists the address ranges during DMA access, effectively overlaying address lines A13–A0.

Table 8–15. Address Ranges for On-Chip Single-Access RAM During External DMA

Device	SARAM (words)	Address Bus	Hex Address Ranges
’C50	9K	A15–A14 ignored, A13–A0 used	0000–2BFF 4000–6BFF 8000–ABFF C000–EBFF
’C51	1K	A15–A14 ignored, A13–A10 must be 0, A9–A0 used	0000–03FF 4000–43FF 8000–83FF C000–C3FF
’C53	3K	A15–A14 ignored, A13–A12 must be 0, A11–A0 used	0000–0BFF 4000–4BFF 8000–8BFF C000–CBFF
’LC56	6K	A15–A14 ignored, A13 must be 0, A12–A0 used	0000–17FF 4000–57FF 8000–97FF C000–D7FF
’C57S/’LC57	6K	A15–A14 ignored, A13 must be 0, A12–A0 used	0000–17FF 4000–57FF 8000–97FF C000–D7FF

DMA access to on-chip single-access RAM is not supported if the device is in concurrent hold mode (HM = 0).

Using DMA on a ’C50 and writing to address 01h affects the second memory location of the SARAM. Furthermore, writing to address 4001h on a ’C50 is equivalent to writing to addresses 01h, 8001h, and C001h, since address lines A14 and A15 are ignored.

Note that the external parallel interface signals are asynchronously disabled during reset; therefore, external DMA is not supported during reset.

8.7 Memory Management

The ’C5x programmable memory map can vary for each application. Instructions are provided for integrating the device memory into the system memory map. The amount and types of memory available on each device are listed in Table 1–1 on page 1-6. Examples of moving and configuring memory are given in this section.

8.7.1 Memory-to-Memory Moves

The following instructions for data and program block moves, word transfers, and the data move function efficiently utilize ’C5x memory spaces.

- Data and program block move instructions
  - BLDD instruction moves a block within data memory
  - BLDP instruction moves a block from data memory to program memory
  - BLPD instruction moves a block from program memory to data memory
- Data and program word transfer instructions
  - The table read (TBLR) instruction reads words from program memory into data memory
  - The table write (TBLW) instruction writes words from data memory to program memory
- Data move (DMOV) instruction allows access to data and operation on that data simultaneously in the same cycle.

For block move instructions, one address is derived from the data address generator, while the other is derived from a long immediate constant or from the BMAR. When used with the repeat instructions (RPT and RPTZ), these instructions efficiently perform block moves from on-chip or off-chip memory.

The DMOV function, implemented in on-chip data RAM, is equivalent to that of the ’C2x. DMOV copies a word from the currently addressed data memory location in on-chip RAM to the next-higher location, while the data from the addressed location is being operated upon in the same cycle (for example, by the CALU). An ARAU operation can also be performed in the same cycle when the indirect addressing mode is used. The DMOV function can implement algorithms that use the z^{-1} delay operation, such as convolution and digital filtering, in which data is passed through a time window.

The DMOV function is most efficient when operating in dual-access on-chip RAM. When operating in single-access RAM, DMOV requires an additional cycle. The DMOV function is contiguous across the boundary of dual-access on-chip RAM blocks B0 and B1. The DMOV function is used by these instructions:

- LTD — load TREG0 and accumulate product with data move
- MACD — multiply and accumulate with data move
- MADD — multiply and accumulate with data move and coefficient address contained in BMAR

Note:
The DMOV operation cannot be performed on external data memory.

8.7.2 Memory Block Moves

The ’C5x devices can address a large amount of off-chip memory but are limited in the amount of on-chip memory. Several instructions can move blocks of data from slower off-chip memories to on-chip memory for faster program execution. In addition, data can be transferred from on-chip to off-chip memory for storage or multiprocessor applications.

8.7.2.1 Moving Data With the BLDD Instruction

The BLDD instruction transfers data in the following ways:

- From external data memory to external data memory
- From external data memory to internal data memory
- From internal data memory to internal data memory
- From internal data memory to external data memory

Example 8–1 illustrates how to use the BLDD instruction to move external data (for example, a table of coefficients) to internal DARAM block B1.

Example 8–1. Moving External Data to Internal Data Memory With the BLDD Instruction

```
*
* This routine uses the BLDD instruction to move external data memory to
* internal data memory.
*
MOVED  LACC   #8000h
       SAMM   BMAR      ;BMAR contains source address in data memory
       LAR    AR7,#300h ;AR7 contains dest. address in data memory
       MAR    *,AR7     ;ARP = AR7
       RPT    #511      ;Move 512 values from data memory to data memory block B1
       BLDD   BMAR,*+
       RET
```

8.7.2.2 Moving Data From Data Memory to Program Memory

The BLDP and TBLW instructions transfer data to program memory in the following ways:

- From external data memory to external program memory
- From external data memory to internal program memory
- From internal data memory to internal program memory
- From internal data memory to external program memory

For systems with external data memory but no external program memory, you can use the BLDP instruction to move additional blocks of code into internal program memory. Example 8–2 illustrates how to use the BLDP instruction to move external data to internal program memory.

You can also use the TBLW instruction to transfer data memory to program memory. The TBLW instruction differs from the BLDP instruction in that the accumulator contains the destination program memory address. This lets you specify a calculated, rather than predetermined, location of a block of data in program memory. Example 8–3 illustrates how to use the TBLW instruction to move external data to internal program memory.

Example 8–2. Moving External Data to Internal Program Memory With the BLDP Instruction

```
*
* This routine uses the BLDP instruction to move external data memory to
* internal program memory. This instruction could be used to boot load a
* program to the on chip program RAM from external data memory.
*
MOVEDP LACC   #2000h
       SAMM   BMAR          ;BMAR contains dest. address in program memory (’C51)
       LAR    AR7,#0F000h   ;AR7 contains source address in data memory
       MAR    *,AR7         ;ARP=AR7
       RPT    #1023         ;Move 1k values from data memory to program memory
       BLDP   *+
       RET
```

Example 8–3. Moving External Data to Internal Program Memory With the TBLW Instruction

```
*
* This routine uses the TBLW instruction to move data memory to program memory.
* The calling routine must contain the destination program memory address in
* the accumulator.
*
TABLEW LAR    AR4,#300h     ;AR4 contains source address in data memory
       MAR    *,AR4         ;ARP = AR4
       RPT    #511          ;Move 512 values from data memory to program memory
       TBLW   *+            ;Accumulator contains dest. address of program memory
       RET
```

8.7.2.3 Moving Data From Program Memory to Data Memory

The BLPD and TBLR instructions transfer program data to data memory in the following ways:

- From external program memory to external data memory
- From external program memory to internal data memory
- From internal program memory to internal data memory
- From internal program memory to external data memory

When no external data memory is available, program memory may contain necessary coefficient tables that should be loaded into internal data memory. Example 8–4 illustrates how to use the BLPD instruction to move external program memory to internal DARAM block B1.

You can also use the TBLR instruction to transfer program data to data memory. The TBLR instruction differs from the BLPD instruction in that the accumulator contains the source program memory address. This lets you specify a calculated, rather than predetermined, location of a block of data in program memory. Example 8–5 illustrates how to use the TBLR instruction to move external program to internal DARAM block B1.

Example 8–4. Moving External Program to Internal Data Memory With the BLPD Instruction

```
*
* This routine uses the BLPD instruction to move external program memory to
* internal data memory. This routine is useful for loading a coefficient
* table stored in external program memory to data memory when no external
* data memory is available.
*
MOVEPD LAR    AR7,#300h     ;AR7 contains dest. address in data memory
       MAR    *,AR7         ;ARP=AR7
       RPT    #127          ;Move 128 values from program memory to data block B1
       BLPD   #0FD00h,*+
       RET
```

Example 8–5. Moving External Program to Internal Data Memory With the TBLR Instruction

```
*
* This routine uses the TBLR instruction to move external program memory to
* internal data memory. The calling routine must contain the source program
* memory address in the accumulator.
*
TABLER LAR    AR3,#300h     ;AR3 contains dest. address in data memory
       MAR    *,AR3         ;ARP=AR3
       RPT    #127          ;Move 128 values from program memory to data block B1
       TBLR   *+            ;Accumulator contains external program memory address
       RET
```

8.7.2.4 Moving Data From Data Memory to I/O Space With the LMMR Instruction

The LMMR instruction can be used to transfer data from external or internal data memory to an external I/O port. Example 8–6 illustrates how to use the LMMR instruction to move data from internal data memory to a memory-mapped I/O port.

Example 8–6. Moving Data From Internal Data Memory to I/O Space With the LMMR Instruction

```
*
* This routine uses the LMMR instruction to move data from internal data memory
* to a memory-mapped I/O port. Note that 16 I/O ports are mapped in data
* page 0 of the 'C5x memory map.
*
OUTPUT:
       LDP    #0            ;DP=0
       RPT    #63           ;Move 64 values from a table beginning at 800h in data
       LMMR   50h,#800h     ;memory to port 50h. Source address is incremented
       RET
```

8.7.2.5 Moving Data From I/O Space to Data Memory With the SMMR Instruction

The SMMR instruction can be used to transfer data from an external I/O port to external or internal data memory. Example 8–7 illustrates how to use the SMMR instruction to move data from a memory-mapped I/O port to internal data memory.

Example 8–7. Moving Data from I/O Space to Internal Data Memory With the SMMR Instruction

```
*
* This routine uses the SMMR instruction to move data from a memory-mapped
* I/O port to internal data memory. Note that 16 I/O ports are mapped in data
* page 0 of the 'C5x memory map.
*
INPUT:
       LDP    #0            :DP=0
       RPT    #511          ;Move 512 values from port 51h to table beginning at
       SMMR   51h,#800h     ;800h in data memory. Dest. address is incremented
       RET
```

8.8 Boot Loader

Several of the ’C5x devices include a boot loader program contained in the on-chip ROM (see Appendix G, *Development Support and Part Order Information*, for part numbering nomenclature). The main function of the boot loader is to transfer code from an external source to the program memory at power-up. This can be done in several different ways, depending on the system requirements. For some applications, a serial interface is appropriate. If the code is already stored in nonvolatile memory (ROM), a parallel interface is more appropriate.

If the MP/$\overline{\text{MC}}$ pin of the ’C5x is sampled low during a hardware reset, program execution begins at address location 0000h of the on-chip ROM. This location contains a branch instruction to the start of the boot-loader program. The on-chip ROM is factory programmed with the boot-loader program. The boot-loader program sets up the CPU status registers before initiating the boot load:

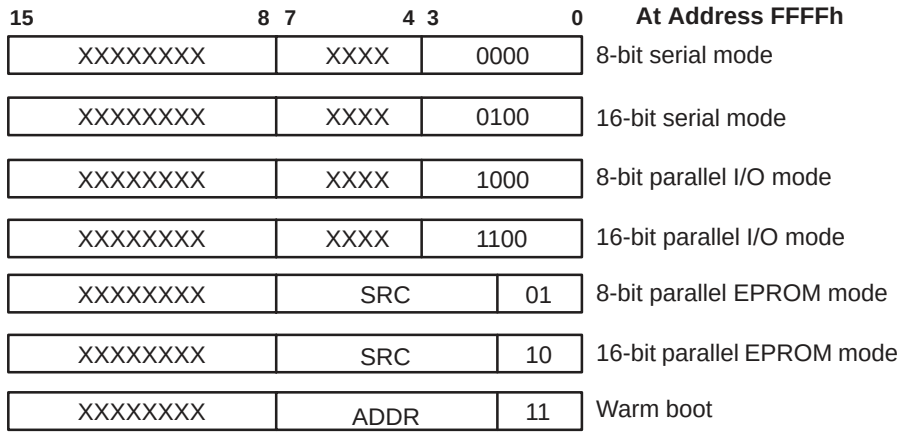
- Interrupts are globally disabled (INTM = 1).
- On-chip DARAM block B0 is mapped into program space (CNF = 1).
- On-chip SARAM block is mapped into program space (RAM = 1, OVLY = 0).
- Entire program and data memory spaces are enabled with seven wait states.
- 32K words of global data memory are enabled initially in data spaces 8000h to FFFFh. After the code transfer is complete, the global memory is disabled before control is transferred to the destination address in program memory.

Note that both DARAM and SARAM memory blocks are enabled in program memory space; this allows you to transfer code to on-chip program memory.

The boot-loader program reads global data memory location FFFFh by driving the bus request ($\overline{\text{BR}}$) and data strobe ($\overline{\text{DS}}$) pins low. The lower 8 bits of the word at address FFFFh specify the boot mode; the higher 8 bits are ignored by the boot loader.

Figure 8–8 lists the available boot mode options and the corresponding values for the boot routine selection word.

Figure 8–8. Boot Routine Selection Word



Legend: X = Don't care condition
SRC = 6-bit page address for parallel EPROM modes
ADDR = 6-bit page address for warm boot mode

8.8.1 HPI Boot Mode ('C57 only)

In HPI boot mode, the boot-loader program first verifies if the host port interface (HPI) boot mode is selected. To select the HPI boot mode, connect the $\overline{\text{HINT}}$ pin to the $\overline{\text{INT3}}$ pin; this sets the $\overline{\text{INT3}}$ bit in the interrupt flag register (IFR) when the $\overline{\text{HINT}}$ pin is asserted low. The boot loader asserts $\overline{\text{HINT}}$ low, waits for 10 CLKOUT1 cycles, and reads the $\overline{\text{INT3}}$ bit. If the $\overline{\text{INT3}}$ bit is set (indicating an $\overline{\text{INT3}}$ interrupt is pending), the boot loader transfers program control to the start address (8800h in program space) of the on-chip HPI RAM and starts executing user code from there. If the $\overline{\text{INT3}}$ bit is not set (indicating that $\overline{\text{HINT}}$ is not connected to $\overline{\text{INT3}}$), the boot loader skips the HPI boot mode and reads the boot routine selection word (Figure 8–8) at global data memory location FFFFh to identify the boot mode.

If the HPI boot mode is selected, the host must download code to the HPI RAM before it brings the 'C5x out of reset. Note that the boot loader keeps HPI in the shared-access mode (SMOD = 1) during the entire boot loading operation. Once $\overline{\text{HINT}}$ is asserted low by the boot loader, $\overline{\text{HINT}}$ remains low until a host controller (if any) clears $\overline{\text{HINT}}$ by writing to the host port interface control register (HPIC).

Instead of connecting the $\overline{\text{HINT}}$ pin to the $\overline{\text{INT3}}$ pin, you can send a valid interrupt to the $\overline{\text{INT3}}$ pin within 30 CLKOUT1 cycles after the 'C5x fetches the reset vector. For 'C5x reset vector fetch timing specifications, refer to the TMS320C5x data sheet.

An alternative to the HPI boot mode is the warm boot mode described in subsection 8.8.5 on page 8-37. The warm boot mode may be preferred to the HPI boot mode, if it is not convenient to connect the $\overline{\text{HINT}}$ pin to the $\overline{\text{INT3}}$ pin or if the program has already been transferred to program memory.

8.8.2 Serial Boot Mode

To select the serial boot mode, the serial port control register (SPC) is set to 00F8h for a16-bit word transfer or to 00FCh for an 8-bit word transfer. See subsection 9.7.1, *Serial Port Interface Registers*, on page 9-24 for a description of each SPC bit.

The external flag (XF) pin signals that the ’C5x is ready to respond to the serial port receive section. The XF pin is set high at reset and is driven low to initiate reception. No frame sync pulses should appear on the FSR pin before XF goes low. The receive clock must be supplied by an external device to the ’C5x.

8.8.2.1 16-Bit Word Serial Transfer

If the 16-bit word transfer is selected (Figure 8–9), the first 16-bit word received by the device from the serial port specifies the destination address (Destination_{16}) of code in program memory. The next 16-bit word specifies the length (Length_{16}) of the actual code that follows. These two 16-bit words are followed by N number of code words to be transferred to program memory. Note that the number of 16-bit words specified by the parameter N does not include the first two 16-bit words received (Destination_{16} and Length_{16}). After the specified number of code words are transferred to program memory, the ’C5x branches to the destination address. The length N is defined as:

length N = number of 16-bit words – 1

Figure 8–9. 16-Bit Word Transfer

Legend:

Destination_{16}	16-bit destination address
Length_{16}	16-bit word that specifies the length of the code (N) that follows
Code Word(N)_{16}	N number of 16-bit words to be transferred

8.8.2.2 8-Bit Word Serial Transfer

If the 8-bit word transfer is selected (Figure 8–10), a higher-order byte and a lower-order byte form a 16-bit word. The first 16-bit word received by the device from the serial port specifies the destination address (Destination_h and Destination_l) of code in program memory. The next 16-bit word specifies the

length ($Length_h$ and $Length_l$) of the actual code that follows. These two 16-bit words are followed by N number of code words to be transferred to program memory. Note that the number of 16-bit words specified by the parameter N does not include the first four bytes (first two 16-bit words) received (Destination and Length). After the specified number of code words are transferred to program memory, the ’C5x branches to the destination address. The length N is defined as:

length N = number of 16-bit words – 1

or

length N = (number of bytes to be transferred ÷ 2) – 1

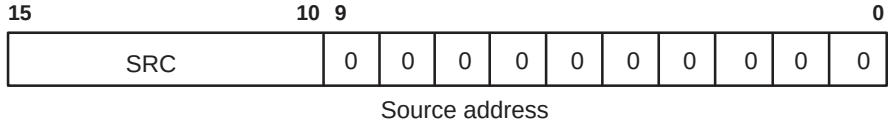
Figure 8–10. 8-Bit Word Transfer

Legend:		
	$Destination_h$	High byte of destination address
	$Destination_l$	Low byte of destination address
	$Length_h$	High byte that specifies the length of the code (N) that follows
	$Length_l$	Low byte that specifies the length of the code (N) that follows
	Code $Word(N)_h$	High byte of N number of 16-bit words to be transferred
	Code $Word(N)_l$	Low byte of N number of 16-bit words to be transferred

8.8.3 Parallel EPROM Boot Mode

The parallel EPROM boot mode is used only when code is stored in EPROMs (8-bit or 16-bit wide). The code is transferred from global data memory (starting at the source address) to program memory (starting at the destination address). The six MSBs of the source address are specified by the SRC field of the boot routine selection word (Figure 8–8 on page 8-33). A 16-bit source address is defined by this SRC field as shown in Figure 8–11. The ’C5x transfers control to the source address after disabling global data memory.

Figure 8–11.16-Bit Source Address for Parallel EPROM Boot Mode



Legend: SRC = 6-bit page address

8.8.3.1 16-Bit Word Parallel Transfer

If the 16-bit word parallel boot mode is selected (Figure 8–9 on page 8-34), boot code will be read in 16-bit words starting at the source address. The

source address is incremented by 1 after every read operation. The first 16-bit word read from the source address specifies the destination address ($Destination_{16}$) of code in program memory. The next 16-bit word specifies the length ($Length_{16}$) of the actual code that follows. These two 16-bit words are followed by N number of code words to be transferred to program memory. Note that the number of 16-bit words specified by the parameter N does not include the first two 16-bit words received ($Destination_{16}$ and $Length_{16}$). After the specified number of code words are transferred to program memory, the 'C5x branches to the destination address. The length N is defined as:

length N = number of 16-bit words – 1

Note that there is at least a 4-instruction-cycle delay between a read from the EPROM and a write to the destination address. This delay ensures that if the destination is in external memory (for example, fast SRAM), there is enough time to turn off the source memory (for example, EPROM) before the write operation is performed.

8.8.3.2 8-Bit Word Parallel Transfer

If the 8-bit word parallel boot mode is selected (Figure 8–10 on page 8-35), two consecutive memory locations (starting at the source address) are read to form a 16-bit word. The high-order byte of the 16-bit word is followed by the low-order byte. Data is read from the lower eight data lines, ignoring the higher byte on the data bus. The first 16-bit word specifies the destination address ($Destination_h$ and $Destination_l$) of code in program memory. The next 16-bit word specifies the length ($Length_h$ and $Length_l$) of the actual code that follows. These two 16-bit words are followed by N number of code words to be transferred to program memory. Note that the number of 16-bit words specified by the parameter N does not include the first four bytes (first two 16-bit words) received (Destination and Length). After the specified number of code words are transferred to program memory, the 'C5x branches to the destination address. The length N is defined as:

length N = number of 16-bit words – 1

or

length N = (number of bytes to be transferred ÷ 2) – 1

Note that there is at least a 4-instruction-cycle delay between a read from the EPROM and a write to the destination address. This delay ensures that if the destination is in external memory (for example, fast SRAM), there is enough time to turn off the source memory (for example, EPROM) before the write operation is performed.

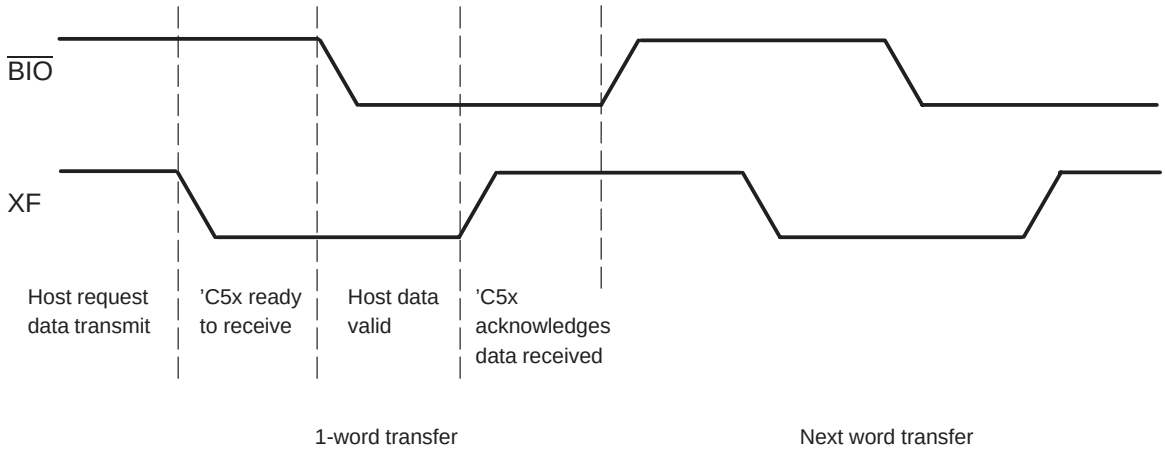
8.8.4 Parallel I/O Boot Mode

The parallel I/O boot mode asynchronously transfers code from the I/O port at address 50h to internal or external program memory. Each word can be 16 bits or 8 bits long. The ’C5x communicates with the external device via the $\overline{\text{BIO}}$ and XF lines. This allows a slow host processor to communicate easily with the ’C5x by polling/driving the $\overline{\text{BIO}}$ and XF lines. The handshake protocol shown in Figure 8–12 on page 8-37 must be used to successfully transfer each word via I/O port 50h.

If the 8-bit boot mode is selected, two consecutive 8-bit words are read to form a 16-bit word. The high-order byte of the 16-bit word is followed by the low-order byte. Data is read from the lower eight data lines of I/O port 50h, ignoring the higher byte on the data bus. For both the 8-bit and 16-bit parallel I/O boot modes, refer to subsection 8.8.3, *Parallel EPROM Boot Mode*, for the description of destination and length code words.

Note that there is at least a 4-instruction-cycle delay between the XF rising edge and a write operation to the destination address. This delay ensures that if the destination is in external memory (for example, fast SRAM), the host processor has enough time to turn off the data buffers before the write operation is performed. The ’C5x accesses the external bus only when XF is high.

Figure 8–12. Handshake Protocol

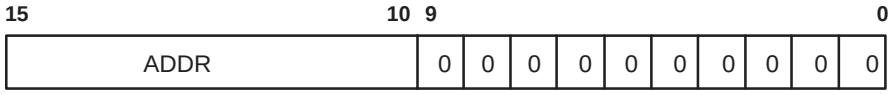


8.8.5 Warm Boot Mode

In a warm boot, the boot loader runs but does not move any code. Control is simply transferred to the entry address. The warm boot mode can be used if the program has already been transferred to internal or external memory by

other means (for example, HPI or external DMA) or if only a warm device reset is required. The six MSBs of the entry address are specified by the ADDR field of the boot routine selection word (Figure 8–8 on page 8-33). A 16-bit entry address is defined by this ADDR field as shown in Figure 8–13. The ’C5x transfers control to the entry address after disabling global data memory. For ’C57 devices, the warm boot mode can be used instead of the HPI boot mode to transfer control to the on-chip HPI RAM.

Figure 8–13. 16-Bit Entry Address for Warm Boot Mode



Entry address

Legend: ADDR = 6-bit page address

8.9 External Parallel Interface Operation

All bus cycles comprise integral numbers of CLKOUT1 cycles. One CLKOUT1 cycle is defined to be from one falling edge of CLKOUT1 to the next falling edge of CLKOUT1. For full-speed, 0-wait-state operation, reads require one cycle. A write immediately preceded by a read or immediately followed by a read requires three cycles. Refer to Figure 8–14 on page 8-40, Figure 8–15 on page 8-41, and Figure 8–16 on page 8-41 for timings for both read and write cycles.

For read cycles, $\overline{\text{STRB}}$ goes low and ADDRESS becomes valid with the falling edge of CLKOUT1. For 0-wait-state read cycles, the $\overline{\text{RD}}$ signal goes low with the rising edge of CLKOUT1 and then goes high at the next falling edge of CLKOUT1. For 1-wait-state (multicycle) read cycles, the $\overline{\text{RD}}$ stays low but goes high with the falling edge of CLKOUT1 before the next cycle, even if the cycles are contiguous. Read data is sampled at the rising edge of $\overline{\text{RD}}$.

The R/$\overline{\text{W}}$ signal goes high at least one half cycle of CLKOUT1 before any read cycle; for contiguous read cycles, $\overline{\text{STRB}}$ stays low. At the end of a read cycle or sequence of reads, $\overline{\text{STRB}}$ and $\overline{\text{RD}}$ go high on the falling edge of CLKOUT1.

Write cycles always have at least one inactive (pad) cycle of CLKOUT1 before and after the actual write operation, including contiguous writes. This allows a smooth transition between the write and any adjacent bus operations or other writes. For this pad cycle, $\overline{\text{STRB}}$ and $\overline{\text{WE}}$ are always high. The R/$\overline{\text{W}}$ signal always changes state on the rising edge of CLKOUT1 during the pad cycle before and after a write or series of writes. This prevents bus contention during a transition between read and write operations. Note that for a series of writes, R/$\overline{\text{W}}$ stays low.

Timing of valid addresses for writes differs, depending on what activities occur before and after the write. Between writes, and for the first and last write in a series, ADDRESS becomes valid on the rising edge of CLKOUT1. If a read immediately follows a write or series of writes, ADDRESS becomes valid for that read cycle one half cycle of CLKOUT1 early — that is, on the rising edge, rather than on the falling edge, of CLKOUT1. This is an exception to the usual read cycle address timing.

For the actual write operation, $\overline{\text{STRB}}$ and $\overline{\text{WE}}$ both go low on the falling edge of CLKOUT1 and stay low until the next falling edge of CLKOUT1 (for 0-wait-state write cycles). For 1-wait-state (multicycle) writes, $\overline{\text{STRB}}$ and $\overline{\text{WE}}$ remain low but go high again on the falling edge of CLKOUT1 at the beginning of the pad cycle. *Write data* is driven approximately at the falling edge of $\overline{\text{STRB}}$ and $\overline{\text{WE}}$ and is held for approximately one half cycle of CLKOUT1 after $\overline{\text{STRB}}$ and $\overline{\text{WE}}$ go high (refer to the TMS320C5x data sheet for actual timing specifications).

Transitions on the external parallel interface control outputs (CLKOUT1, $\overline{\text{STRB}}$, $\overline{\text{WE}}$, and $\overline{\text{RD}}$) are all initiated by the same two internal clocks. Since these signals also use the same output buffer circuitry, they all switch within close tolerances of each other, as specified in the TMS320C5x data sheet.

Transitions on the address bus and other related outputs ($\overline{\text{IS}}$, $\overline{\text{PS}}$, $\overline{\text{DS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$) are initiated by the same internal signals that cause transitions on the control outputs; however, the internal device logic that generates these outputs is different from the circuitry used for the control outputs. Therefore, transitions on the address bus and related outputs typically occur later than control-line transitions.

Timings of control outputs with respect to CLKOUT1 are specified in the TMS320C5x data sheet. Address timings with respect to CLKOUT1 can be derived from address timings for control signals and control signal timings for CLKOUT1. For example, the delay from CLKOUT1 falling to address bus valid at the beginning of a read cycle is calculated as:

[H – (address setup to $\overline{\text{RD}}$)] + maximum positive $\overline{\text{RD}}$ to CLKOUT1 skew
(refer to the TMS320C5x data sheet for specific timing values)

Other interface timings with respect to CLKOUT1 can be calculated in the same manner.

Figure 8–14. External Interface Operation for Read-Read-Write (Zero Wait States)

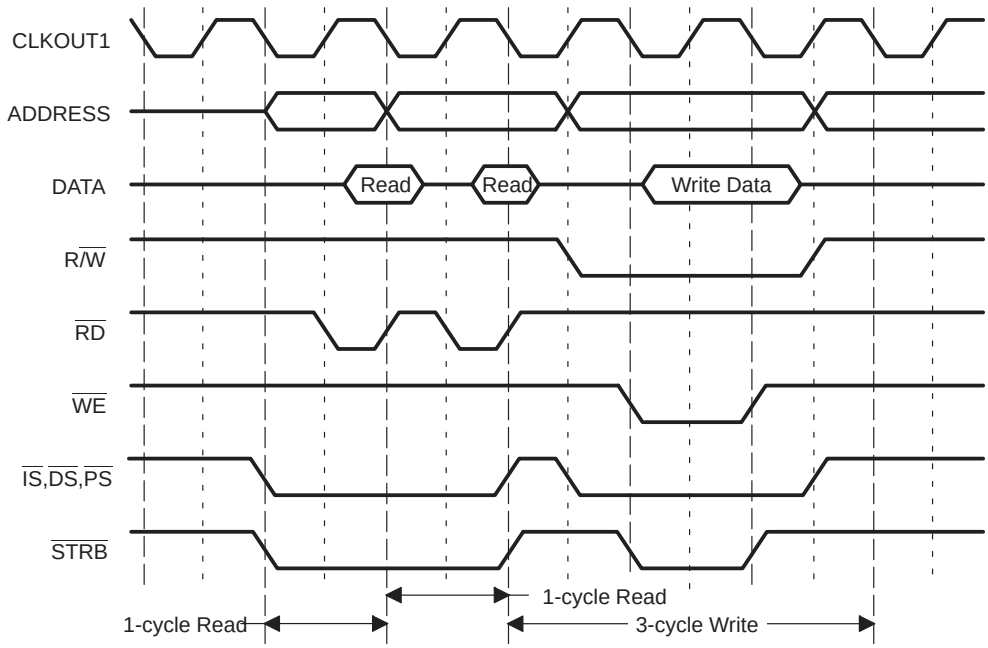


Figure 8–15. External Interface Operation for Write-Write-Read (Zero Wait States)

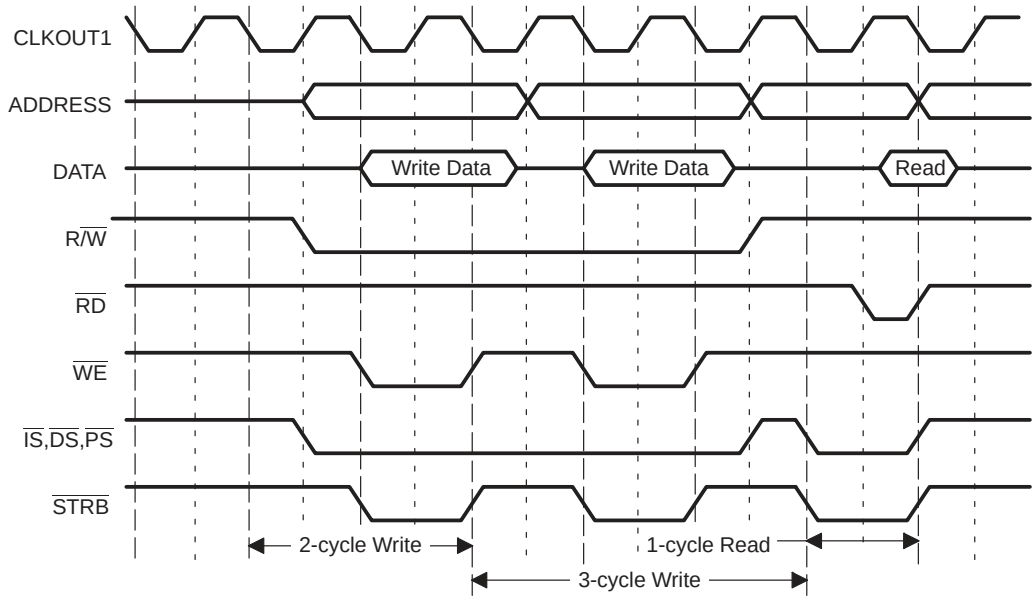
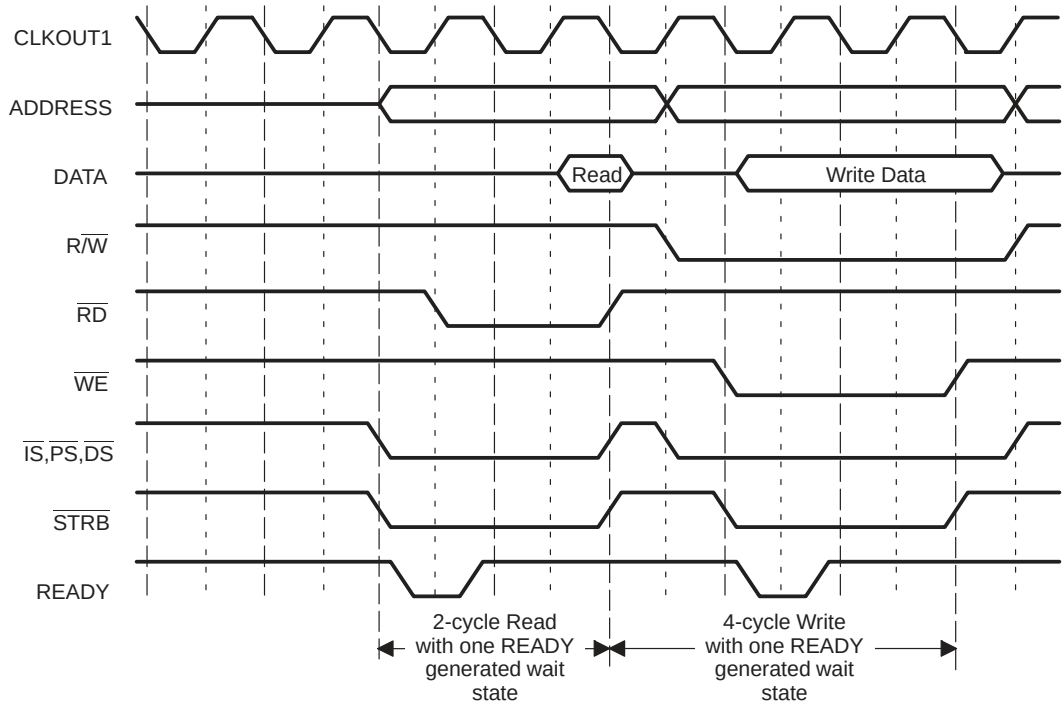


Figure 8–16. External Interface Operation for Read-Write (One Wait State)



8.10 Software Wait-State Generation

The software-programmable wait-state generators can be used to extend external bus cycles by up to seven machine cycles. All external reads require at least one machine cycle, while all external writes require at least two machine cycles. However, as shown in Figure 8–14 and Figure 8–15, an external write immediately followed or immediately preceded by an external read cycle requires three cycles. This provides a convenient way for interfacing external devices that do not satisfy the full-speed access-time requirements of the ’C5x. The ’C5x can generate wait states to extend the memory read/write cycles by software-programmable wait-state generators or by an interface with the hardware READY line. The software-programmable wait-state generators can only generate up to seven wait states. External devices requiring more than seven wait states can use the hardware READY line to generate the wait states.

Note that if the on-chip wait-state generator is used to add wait states for external accesses, the number of CLKOUT1 cycles required for writes is not effected until two or more wait states are specified, contrary to wait states generated with the external READY input. Table 8–16 shows the number of cycles required for the different types of external device accesses.

Table 8–16. Number of CLKOUT1 Cycles Per Access for Various Numbers of Wait States

Number of Wait States	Number of CLKOUT1 Cycles†			
	Hardware Wait State		Software Wait State	
	Read	Write	Read	Write
0	1	2n + 1	1	2n + 1
1	2	3n + 1	2	2n + 1
2	3	4n + 1	3	3n + 1
3	4	5n + 1	4	4n + 1

† Where n is the number of consecutive write cycles.

Also, note that the external READY input is sampled only after the internal software wait states are completed. Therefore, if the READY input is driven low before the completion of the internal software wait states, no wait states are added to the external memory access until the specified number of software wait states is completed. Wait states are only added if the READY input is still low after the software wait states are completed. Additionally, it should be noted that the READY input is not an asynchronous input and input setup and hold times for this signal as specified in the TMS320C5x data sheet must be met or significant device malfunction will result.

Chapter 9

On-Chip Peripherals

The on-chip peripheral interfaces connected to the ’C5x CPU include the divide-by-one clock, timer, software-programmable wait-state generators, general purpose I/O pins, parallel I/O ports, serial ports, and host port interface. These peripherals are controlled through registers that reside in the memory map. The serial ports and timer are synchronized to the processor via interrupts.

Topic **Page**

9.1 Peripheral Control

Peripheral circuits are operated and controlled through access of memory-mapped control and data registers. The operation of the serial ports and the timer is synchronized to the processor via interrupts or through interrupt polling. Setting and clearing bits can enable, disable, initialize, and dynamically reconfigure the peripherals. Data is transferred to and from the peripherals through memory-mapped data registers. When a peripheral is not in use, the internal clocks can be shut off from that peripheral, allowing for lower power consumption when the device is in normal run mode or idle mode.

9.1.1 Memory-Mapped Peripheral Registers and I/O Ports

There are 28 processor registers, 17 peripheral registers, and 16 I/O ports mapped into the data memory space. Table 9–1 lists the memory-mapped registers and I/O ports of the ’C5x. Note that all writes to memory-mapped peripheral registers (but not processor registers or memory-mapped I/O ports) require one additional CLKOUT1 cycle.

Table 9–1. Data Page 0 Address Map — Peripheral Registers and I/O Ports

Address			
Dec	**Hex**	**Name**	**Description**
0–3	0–3	——	Reserved
4–31	4–1F	——	Memory-mapped processor registers (see subsection 8.3.2, *Local Data Memory Address Map*, on page 8-17).
32	20	DRR	Data receive register
33	21	DXR	Data transmit register
34	22	SPC	Serial port control register
35	23	——	Reserved
36	24	TIM	Timer counter register
37	25	PRD	Timer period register
38	26	TCR	Timer control register
39	27	——	Reserved
40	28	PDWSR	Program/data wait-state register
41	29	IOWSR	I/O port wait-state register
42	2A	CWSR	Wait-state control register

Table 9–1. Data Page 0 Address Map — Peripheral Registers and I/O Ports (Continued)

Address			
Dec	**Hex**	**Name**	**Description**
43–47	2B–2F	——	Reserved for test/emulation
48	30	TRCV	TDM data receive register
		BDRR	BSP data receive register
49	31	TDXR	TDM data transmit register
		BDXR	BSP data transmit register
50	32	TSPC	TDM serial port control register
		BSPC	BSP control register
51	33	TCSR	TDM channel select register
		SPCE	BSP control extension register
52	34	TRTA	TDM receive/transmit address register
		AXR	BSP address transmit register
53	35	TRAD	TDM receive address register
		BKX	BSP transmit buffer size register
54	36	ARR	BSP address receive register
55	37	BKR	BSP receive buffer size register
56–79	38–4F	——	Reserved
80	50	PA0	I/O port 50h
81	51	PA1	I/O port 51h
82	52	PA2	I/O port 52h
83	53	PA3	I/O port 53h
84	54	PA4	I/O port 54h
85	55	PA5	I/O port 55h
86	56	PA6	I/O port 56h
87	57	PA7	I/O port 57h
88	58	PA8	I/O port 58h
89	59	PA9	I/O port 59h

Table 9–1. Data Page 0 Address Map — Peripheral Registers and I/O Ports (Continued)

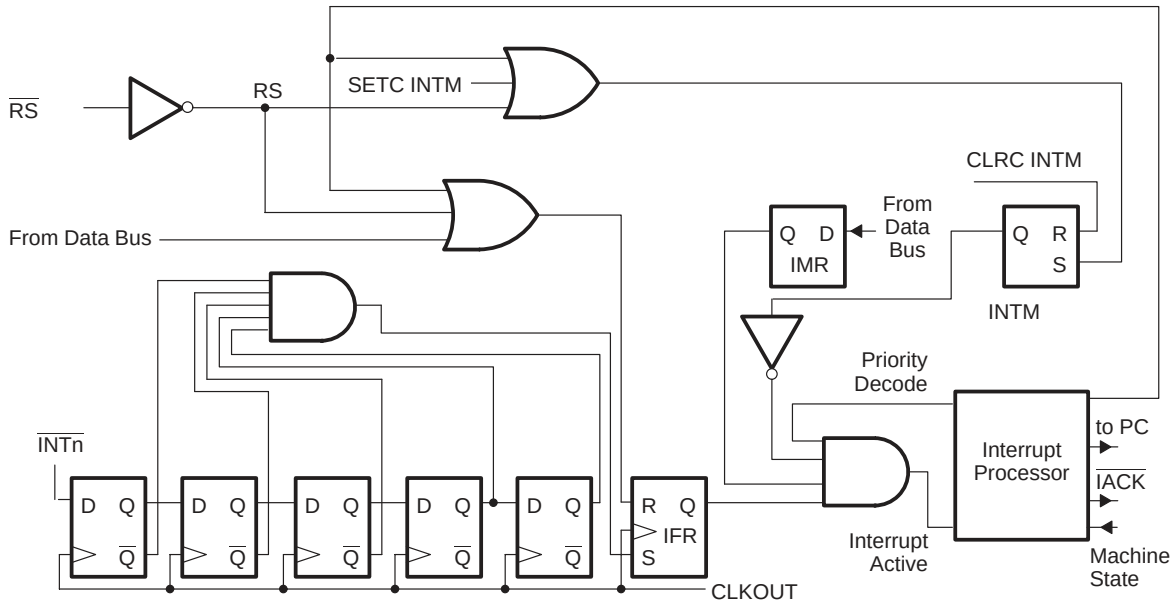
Address			
Dec	**Hex**	**Name**	**Description**
90	5A	PA10	I/O port 5Ah
91	5B	PA11	I/O port 5Bh
92	5C	PA12	I/O port 5Ch
93	5D	PA13	I/O port 5Dh
94	5E	PA14	I/O port 5Eh
95	5F	PA15	I/O port 5Fh
96–127	60–7F	——	Scratch-pad RAM (DARAM B2)

9.1.2 External Interrupts

The ’C5x has four external, maskable user interrupts ($\overline{\text{INT4}}$–$\overline{\text{INT1}}$) that external devices can use to interrupt the processor, and one external nonmaskable interrupt ($\overline{\text{NMI}}$). Internal interrupts are generated by the timer (TINT), the serial port (RINT, XINT, TRNT, TXNT, BRNT, and BXNT), the host port (HINT), and the software interrupt instructions (TRAP, NMI and INTR). Interrupt priorities are set so that reset ($\overline{\text{RS}}$) has the highest priority and $\overline{\text{INT4}}$ has the lowest priority. The $\overline{\text{NMI}}$ has the second highest priority. For further information regarding interrupt operation, see Section 4.8, *Interrupts*, on page 4-36.

Interrupts may be asynchronously triggered. In the functional logic organization for $\overline{\text{INT4}}$–$\overline{\text{INT1}}$, shown in Figure 9–1, the external interrupt INTn is synchronized to the core via a five flip-flop synchronizer. The actual implementation of the interrupt circuits is similar to this logic implementation. If a 1-1-0-0-0 sequence on five consecutive CLKOUT1 cycles is detected, a 1 is loaded into the interrupt flag register (IFR).

Figure 9–1. External Interrupt Logic Diagram



The ’C5x devices sample the external interrupt pins multiple times to avoid noise-generated interrupts. To detect an active interrupt, the ’C5x must sample the signal low on at least three consecutive machine cycles. Once an interrupt is detected, the ’C5x must sample the signal high on at least two consecutive machine cycles to be able to detect another interrupt. The external interrupt pins are sampled on the rising edge of CLKOUT1. If the external interrupts are running asynchronously, the pulses should be stretched to guarantee three consecutive low samples. Note that if the CPU is in IDLE2 mode, an interrupt input must be high for at least four CLKOUT1 cycles and low for a minimum of five CLKOUT1 cycles to be properly recognized.

If the INTM bit and mask register are properly enabled, the interrupt signal is accepted by the processor. INTM is set and the appropriate IFR bit is cleared when the INTR instruction is jammed into the pipeline, and then after three CLKOUT1 cycles, $\overline{\text{IACK}}$ is generated (see Figure 4–9 on page 4-44). $\overline{\text{NMI}}$ uses the same logic as for $\overline{\text{INT1}}$–$\overline{\text{INT4}}$, except that $\overline{\text{NMI}}$ is not affected by the status of the interrupt mask register (IMR) or the INTM bit.

9.1.3 Peripheral Reset

A number of actions occur when the ’C5x is reset. Section 4.9, *Reset*, on page 4-45 describes the events that occur when the ’C5x is reset. On a device reset, the central processing unit (CPU) sends an $\overline{\text{SRESET}}$ signal to the peripheral circuits. The $\overline{\text{SRESET}}$ signal affects the peripheral circuits in the following ways:

1) The two software wait-state registers (IOWSR and PDWSR) are set to FFFFh, causing all external accesses to occur with seven wait states. The CWSR is loaded with 0Fh.

2) The FO bits of the SPC and TSPC/BSPC are cleared, which selects a word length of 16 bits for each serial port.

3) The FSM bits of the SPC and TSPC/BSPC are cleared. The FSM bit must be set for operation with frame sync pulses.

4) The TXM bits of the SPC and TSPC/BSPC are cleared, which configures the FSX and TFSX pins as inputs.

5) The SPC and TSPC/BSPC are loaded with 0y00h, where the two MSBs of y are 10_2 and the two LSBs of y reflect the current levels on the transmit and receive clock pins of the respective port.

6) The TIM and PRD are loaded with FFFFh. The TDDR and TSS fields of the TCR are cleared and the timer starts.

7) On the HPI, HINT and SMOD are cleared while in reset, and then set after reset goes high.

Refer to Section 4.9 for further details of reset operation.

9.2 Clock Generator

The ’C5x clock generator consists of an internal oscillator and a phase lock loop (PLL) circuit that provides the flexibility for the system designer to select the clock source. The clock generator is driven by a crystal resonator circuit or by an external clock source.

9.2.1 Standard Clock Options (’C50, ’C51, ’C52, ’C53, and ’C53S only)

Table 9–2 lists the standard clock options available. When the internal divide-by-2 option is selected, the internal oscillator is enabled by connecting a crystal across the X1 and X2/CLKIN pins. The frequency of CLKOUT1 is one-half the crystal oscillating frequency. When the external divide-by-2 option is selected, the external clock source is connected directly to the X2/CLKIN pin and the X1 pin is unconnected. The external frequency is divided by two to generate the internal machine cycle.

When the PLL option is selected, the external clock source is connected directly to the CLKIN2 pin, the X1 pin is disconnected from V_{DD}, and the X2/CLKIN pin is connected to V_{DD}. For the ’C50, ’C51, ’C53, and ’C53S, the external frequency is multiplied by one to generate the internal machine cycle. For the ’C52, the external frequency is multiplied by two to generate the internal machine cycle.

Table 9–2. Standard Clock Options (’C50, ’C51, ’C52, ’C53, and ’C53S only)

CLKMD1	CLKMD2	Clock Mode
0	0	External divide-by-2 option with internal oscillator disabled.
0	1	Reserved for test purposes.
1	0	PLL clock generator option. ❑ For ’C50, ’C51, ’C53, and ’C53S: multiply-by-1 option ❑ For ’C52: multiply-by-2 option
1	1	External divide-by-2 option or internal divide-by-2 option with an external crystal.

9.2.2 PLL Clock Options (’LC56, ’C57S, and ’LC57 only)

Table 9–3 lists the PLL clock options available. The PLL circuit provides the capability to supply lower external frequency sources than the machine cycle rate of the CPU. This is a desirable feature because it reduces a system’s high-frequency noise that is due to a high-speed switching clock. When the PLL option is selected, the external clock source is connected directly to the X2/CLKIN pin.

The PLL has a maximum operating frequency of 28.6 MHz (on a 35-ns ’C5x device). The PLL requires a transitory locking time which is specified in the TMS320C5x data sheet. When the device is in idle2 power-down mode or in stop mode, the PLL stops; in idle power-down mode, the PLL continues operating. See the TMS320C5x data sheet for more information on the external input frequency specification.

Note that the clock mode should not be reconfigured with the clock mode pins during the normal operation. During the idle2 mode, the clock mode can be reconfigured after CLKOUT1 settling high.

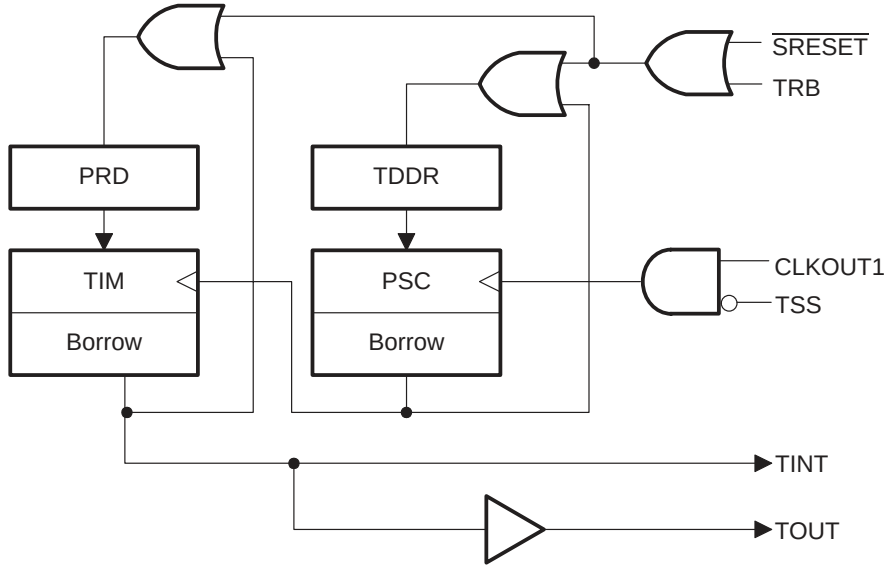
Table 9–3. PLL Clock Options (’LC56, ’C57S, and ’LC57 only)

CLKMD1	CLKMD2	CLKMD3	Clock Mode
0	0	0	PLL multiply-by-3 option
0	0	1	External divide-by-2 option with internal oscillator disabled
0	1	0	PLL multiply-by-4 option
0	1	1	PLL multiply-by-2 option
1	0	0	PLL multiply-by-5 option
1	0	1	PLL multiply-by-1 option
1	1	0	PLL multiply-by-9 option
1	1	1	External divide-by-2 option or internal divide-by-2 option with an internal oscillator enabled

9.3 Timer

The timer is an on-chip down counter that can be used to periodically generate CPU interrupts. Figure 9–2 shows a logical block diagram of the timer. The timer is driven by a prescaler which is decremented by 1 at every CLKOUT1 cycle. A timer interrupt (TINT) is generated each time the counter decrements to 0. The timer provides a convenient means of performing periodic I/O or other functions. When the timer is stopped (TSS = 1), the internal clocks to the timer are shut off, allowing the circuit to run in a low-power mode of operation.

Figure 9–2. Timer Block Diagram



9.3.1 Timer Registers

The timer operation is controlled via the timer control register (TCR), the timer counter register (TIM), and the timer period register (PRD). Figure 9–3 shows and Table 9–4 describes the TCR bit fields.

Figure 9–3. Timer Control Register (TCR) Diagram

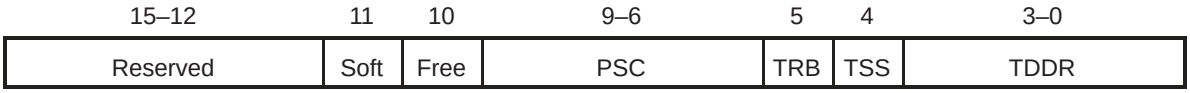


Table 9–4. Timer Control Register (TCR) Bit Summary

Bit	Name	Reset value	Function
15–12	Reserved	—	These bits are reserved and are always read as 0.
11	Soft	0	This bit is used in conjunction with the Free bit to determine the state of the timer when a halt is encountered. When the Free bit is cleared, the Soft bit selects the emulation mode. Soft = 0 The timer stops immediately. Soft = 1 The timer stops after decrementing to zero.
10	Free	0	This bit is used in conjunction with the Soft bit to determine the state of the timer when a halt is encountered. When the Free bit is cleared, the Soft bit selects the emulation mode. Free = 0 The Soft bit selects the timer mode. Free = 1 The timer runs free regardless of the Soft bit.
9–6	PSC	—	Timer prescaler counter bits. These bits specify the count for the on-chip timer. When the PSC is decremented past 0 or the timer is reset, the PSC is loaded with the contents of the TDDR, and the TIM is decremented.
5	TRB	—	Timer reload bit. This bit resets the on-chip timer. When the TRB is set, the TIM is loaded with the value in the PRD and the PSC is loaded with the value in the TDDR. The TRB is always read as a 0.
4	TSS	0	Timer stop status bit. This bit stops or starts the on-chip timer. At reset, the TSS bit is cleared and the timer immediately starts timing.Note that due to timer logic implementation, two successive writes of one to the TSS bit are required to properly stop the timer. TSS = 0 The timer is started. TSS = 1 The timer is stopped.
3–0	TDDR	0000	Timer divide-down register bits. These bits specify the timer divide-down ratio (period) for the on-chip timer. When the PSC bits are decremented past 0, the PSC is loaded with the contents of the TDDR.

9.3.2 Timer Operation

When the PSC decrements to 0 or when the timer is reset by setting the TRB bit, the contents of the TDDR are loaded into the PSC and the TIM is decremented.

When the TIM decrements to 0 or when the timer is reset by setting the TRB bit, the contents of the PRD are loaded into the TIM. The TRB bit is always read as 0. When a 1 is written to the TRB, the timer is reset, but TRB is still read as 0.

Note:

The current value in the timer can be read by reading the TIM; the PSC can be read by reading the TCR. Because it takes two instructions to read both registers, there may be a change between the two reads as the counter decrements. Therefore, when making precise timing measurements, it may be more accurate to stop the timer to read these two values. Due to timer logic implementation, two instructions are also required to properly stop the timer; therefore, two successive writes of one to the TSS bit should be made when the timer must be stopped.

The timer interrupt (TINT) rate is given by:

$$\text{TINT rate} = \frac{1}{t_{c(C)} \times u \times v} = \frac{1}{t_{c(C)} \times (\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

where $t_{C(C)}$ is the period of CLKOUT1, u is the sum of the TDDR contents + 1, and v is the sum of the PRD contents + 1.

The TINT rate equals the CLKOUT1 frequency divided by two independent factors. The two divisors are implemented with a down counter and period register (see Figure 9–2 on page 9-9) in each stage. The PSC and TDDR fields of the TCR are used for the first stage and the TIM and PRD are used for the second stage. Each time a down counter (PSC or TIM) decrements to 0, a borrow is generated on the next CLKOUT1 cycle, and the down counter is reloaded with the contents of its corresponding period register (TDDR or PRD). The output of the second stage is the TINT signal sent to the CPU and to the timer output (TOUT) pin. The width of the borrow pulse that appears on the output of the second stage equals $t_{C(C)}$.

The timer can be used to generate a sample clock for an analog interface. Example 9–1 uses the timer to generate a sample rate of 50 kHz. Consider an analog-to-digital converter operating at this sample rate. Example 9–2 shows a typical interrupt service routine (ISR).

Example 9–1. Code Initialization for Generating a 50-kHz Clock Signal

```
*Clkin frequency = 20 MHz, timer is running at 10 MHz.
*
   LDP    #0
   SPLK   #199,PRD      ;Load timer period for 20 us period.
   OPL    #8,IMR        ;Set timer interrupt mask bit
   SPLK   #20h,TCR      ;reload and start timer.
   SPLK   #1000b,IFR    ;Clear any pending timer interrupts.
   CLRC   INTM          ;global interrupt enable.
*
```

Example 9–2. Interrupt Service Routine for a 50-kHz Sample Rate

```
*50 kHz sample rate A/D interrupt service routine
*
TIMER_ISR MAR*,AR3  ;Use auxiliary register reserved for
                    ;timer ISR.
          IN  *,14  ;Read A/D.
          RETE      ;Re–enable interrupts and return.
*
```

9.4 Software-Programmable Wait-State Generators

The software-programmable wait-state generators can extend external bus cycles by up to seven machine cycles. This operation provides a convenient means to interface the ’C5x to external devices that do not satisfy the full-speed access-time requirement of the ’C5x. Devices that require more than seven wait states can be interfaced using the hardware READY line. When all external accesses are configured for zero wait states, the internal clocks to the wait-state generators are shut off; shutting off the internal clocks allows this circuitry to run with lower power consumption.

Note:

The wait-state generators affect external accesses only.

Two 16-bit wait-state registers and a 5-bit control register control the software-programmable wait-state generators. Each of the three external spaces (program, data, and I/O spaces) has an assigned field in a software wait-state register.

9.4.1 Program/Data Wait-State Register (PDWSR)

The program and data memory spaces each consist of 64K word addresses. You can view each 64K-word space as being composed of four 16K-word blocks. Each 16K-word block in program and data space is associated with a 2-bit wait-state field in the PDWSR, as shown in Figure 9–4 and listed in Table 9–5. The value of the 2-bit field in PDWSR specifies the number of wait states to be inserted for each access in the given address range. At reset, the PDWSR is set to FFFFh.

Figure 9–4. Program/Data Wait-State Register (PDWSR) Diagram (’C50, ’C51, and ’C52 only)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Data 4	Data 3	Data 2	Data 1	Program 4	Program 3	Program 2	Program 1

Table 9–5. Program/Data Wait-State Register (PDWSR) Address Ranges ('C50, 'C51, and 'C52 only)

PDWSR Bits	Memory Space	Hex Address Range
15–14	Data 4	C000–FFFF
13–12	Data 3	8000–BFFF
11–10	Data 2	4000–7FFF
9–8	Data 1	0000–3FFF
7–6	Program 4	C000–FFFF
5–4	Program 3	8000–BFFF
3–2	Program 2	4000–7FFF
1–0	Program 1	0000–3FFF

The 'C53S, 'LC56, and 'C57 implement a simpler version of the software wait states. Program, data, and I/O space wait states are specified by a single wait-state value. All external addresses in each space may be independently set from 0 to 7 wait states by the 3-bit wait-state field in the PDWSR, as shown in Figure 9–5 and listed in Table 9–6.

Figure 9–5. Program/Data Wait-State Register (PDWSR) Diagram ('C53S, 'LC56, and 'C57 only)

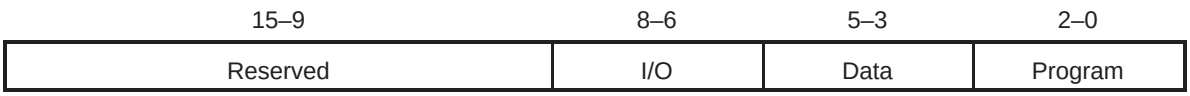


Table 9–6. Program/Data Wait-State Register (PDWSR) Address Ranges ('C53S, 'LC56, and 'C57 only)

Wait-State Field Bits	Space	Hex Address Range
15–9	Reserved	—
8–6	I/O	0000–FFFF
5–3	Data	0000–FFFF
2–0	Program	0000–FFFF

Note that if the on-chip wait-state generator is used to add wait states for external accesses, the number of CLKOUT1 cycles required for writes is not effected until two or more wait states are specified, contrary to wait states generated with the external READY input. Table 9–7 shows the number of cycles required for the different types of external device accesses.

Also, note that the external READY input is sampled only after the internal software wait states are completed. Therefore, if the READY input is driven low before the completion of the internal software wait states, no wait states are added to the external memory access until the specified number of software wait states is completed. Wait states are only added if the READY input is still low after the software wait states are completed. Additionally, it should be noted that the READY input is not an asynchronous input and input setup and hold times for this signal as specified in the TMS320C5x data sheet must be met or significant device malfunction will result.

Table 9–7. Number of CLKOUT1 Cycles per Access for Various Numbers of Wait States

	Number of CLKOUT1 Cycles†			
	Hardware Wait State		Software Wait State	
Number of Wait States	Read	Write	Read	Write
0	1	2n + 1	1	2n + 1
1	2	3n + 1	2	2n + 1
2	3	4n + 1	3	3n + 1
3	4	5n + 1	4	4n + 1

† Where n is the number of consecutive write cycles.

9.4.2 I/O Wait-State Register (IOWSR)

The I/O space consists of 64K word addresses. The IOWSR, shown in Figure 9–6, can be mapped in either of two ways, as specified by the BIG bit in the wait-state control register (CWSR). The value of the 2-bit field in IOWSR specifies the number of wait states to be inserted for each access in the given port or address range (Table 9–8). At reset, the IOWSR is set to FFFFh.

If the BIG bit is cleared, each of eight pairs of memory-mapped I/O ports is associated with a 2-bit wait-state field in IOWSR. The value of the 2-bit field in IOWSR specifies the number of wait states to be inserted for each access in the given port. The entire I/O space is configured with wait states on 2-word boundaries (that is, port 0/1, port 10/11, and port 20/21 all have the same number of wait states). This configuration provides maximum flexibility when I/O bus-cycles access peripherals such as D/A and A/D devices.

If the BIG bit is set, the 64K-word space is divided into eight 8K-word blocks. Each 8K-word block in I/O space is associated with a 2-bit wait-state field in the IOWSR. The value of the 2-bit field in IOWSR specifies the number of wait states to be inserted for each access in the given address range.

Figure 9–6. I/O Port Wait-State Register (IOWSR) Diagram

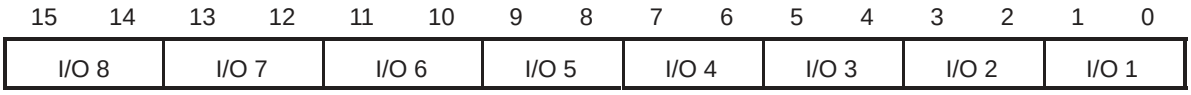


Table 9–8. I/O Port Wait-State Register (IOWSR) Address Ranges

Wait-State Field Bits	I/O Space	Ports/Hex Address Range	
		BIG = 0	BIG = 1
0–1	I/O 1	Port 0/1, port 10/11, etc.	0000–1FFF
2–3	I/O 2	Port 2/3, port 12/13, etc.	2000–3FFF
4–5	I/O 3	Port 4/5, port 14/15, etc.	4000–5FFF
6–7	I/O 4	Port 6/7, port 16/17, etc.	6000–7FFF
8–9	I/O 5	Port 8/9, port 18/19, etc.	8000–9FFF
10–11	I/O 6	Port 0A/0B, port 1A/1B, etc.	A000–BFFF
12–13	I/O 7	Port 0C/0D, Port 1C/1D, etc.	C000–DFFF
14–15	I/O 8	Port 0E/0F, Port 1E/1F, etc.	E000–FFFF

9.4.3 Wait-State Control Register (CWSR)

The CWSR allows you to select one of two mappings of the IOWSR and one of two mappings between 2-bit wait-state fields and the number of wait states for the corresponding space in the PDWSR and IOWSR. The CWSR bit fields are shown in Figure 9–7 and described in Table 9–9. If a bit is cleared, the number of wait states for external accesses in that space is equal to the wait-state field value. If a bit is set, the number of wait states for external accesses in that space is determined by the wait-state field values listed in Table 9–10. Always program the CWSR before configuring the PDWSR and IOWSR to avoid configuring memory with too few wait states during the set-up of wait-state registers.

Figure 9–7. Wait-State Control Register (CWSR) Diagram

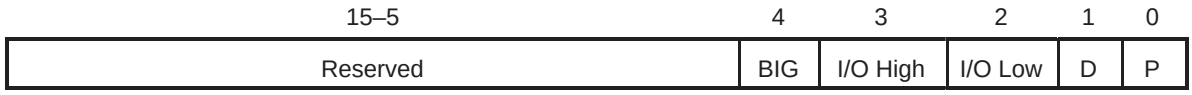


Table 9–9. Wait-State Control Register (CWSR) Bit Summary

Bit	Name	Reset value	Function	
15–5	Reserved	0	These bits are reserved.	
4	BIG	0	This bit specifies how the IOWSR is mapped.	
			BIG = 0	The IOWSR is divided into eight pairs of I/O ports with a 2-bit wait-state field assigned to each pair of ports.
			BIG = 1	The I/O space is divided into eight 8K-word blocks with a 2-bit wait-state field assigned to each block.
3	I/O High	1	This bit is used in conjunction with the 2-bit wait-state field in the IOWSR to determine the number of wait states for the I/O space upper half (I/O 5–I/O 8). See Table 9–10 for the wait state configurations.	
			I/O High = 0	The number of wait states assigned to the I/O space upper half is 0, 1, 2, or 3.
			I/O High = 1	The number of wait states assigned to the I/O space upper half is 0, 1, 3, or 7.

Table 9–9. Wait-State Control Register (CWSR) Bit Summary (Continued)

Bit	Name	Reset value	Function
2	I/O Low	1	This bit is used in conjunction with the 2-bit wait-state field in the IOWSR to determine the number of wait states for the I/O space lower half (I/O 1–I/O 4). See Table 9–10 for the wait state configurations. I/O Low = 0 — The number of wait states assigned to the I/O space lower half is 0, 1, 2, or 3. I/O Low = 1 — The number of wait states assigned to the I/O space lower half is 0, 1, 3, or 7.
1	D	1	Data memory space bit. This bit is used in conjunction with the 2-bit wait-state field in the PDWSR to determine the number of wait states for the data memory space. See Table 9–10 for the wait state configurations. D = 0 — The number of wait states assigned to the data memory space is 0, 1, 2, or 3. D = 1 — The number of wait states assigned to the data memory space is 0, 1, 3, or 7.
0	P	1	Program memory space bit. This bit is used in conjunction with the 2-bit wait-state field in the PDWSR to determine the number of wait states for the program memory space. See Table 9–10 for the wait state configurations. P = 0 — The number of wait states assigned to the program memory space is 0, 1, 2, or 3. P = 1 — The number of wait states assigned to the program memory space is 0, 1, 3, or 7.

Table 9–10. Wait-State Field Values and Number of Wait States as a Function of CWSR Bits 0–3

Wait-State Field of PDWSR or IOWSR†	No. of Wait States (CWSR Bit 0–3 = 0)	No. of Wait States (CWSR Bit 0–3 = 1)
00	0	0
01	1	1
10	2	3
11	3	7

† This bit field corresponds to the wait-state field bits in Figure 9–4 and Figure 9–6.

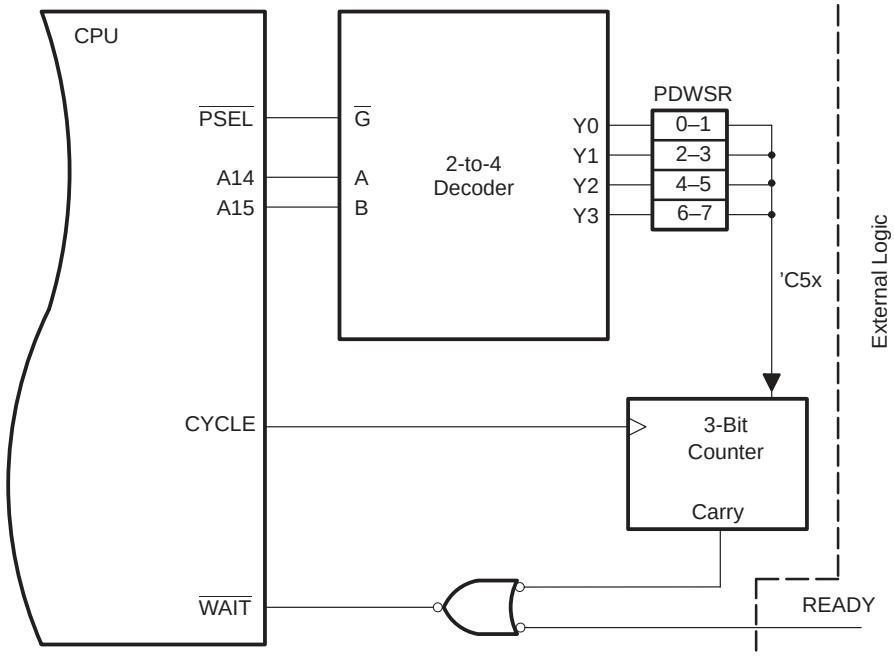
9.4.4 Logic for External Program Space

Figure 9–8 shows a block diagram of the wait-state generator logic for external program space. When an external program access is decoded, the appropriate field of the PDWSR is loaded into the counter. If the field is not 000_2, a not-ready signal is sent to the CPU. The not-ready condition is maintained until the counter decrements to 0 and the external READY line is set high. The external READY and the wait-state READY are ORed to generate the CPU $\overline{\text{WAIT}}$ signal. The READY line is sampled at the rising edge of CLKOUT1.

Note:

The external READY line is sampled only at the last cycle of an external access if the on-chip wait-state generator is used to insert software wait states.

Figure 9–8. Software-Programmable Wait-State Generator Block Diagram



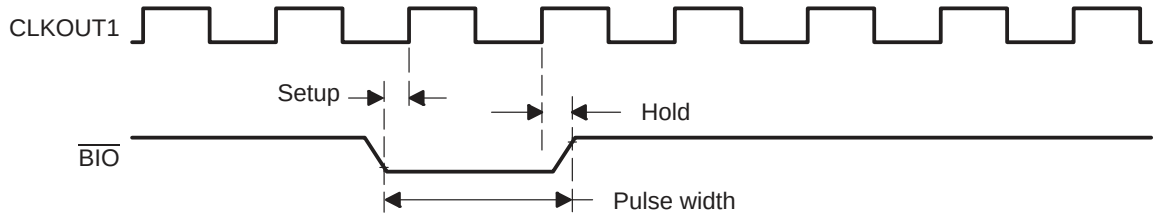
9.5 General-Purpose I/O Pins

The ’C5x has two general-purpose pins that are software controlled. The branch control input ($\overline{\text{BIO}}$) pin and the external flag output (XF) pin. For detailed timing specifications of $\overline{\text{BIO}}$ and XF signals, refer to the TMS320C5x data sheet.

9.5.1 Branch Control Input ($\overline{\text{BIO}}$)

The $\overline{\text{BIO}}$ pin monitors peripheral device status—especially as an alternative to an interrupt when time-critical loops must not be disturbed. A branch can be conditionally executed dependent upon the state of the $\overline{\text{BIO}}$ input. The timing diagram, shown in Figure 9–9, shows the $\overline{\text{BIO}}$ operation (refer to the TMS320C5x data sheet for actual timing specifications). This timing diagram is for a sequence of single-cycle, signal-word instructions located in external memory. When used with the XC instruction, the $\overline{\text{BIO}}$ condition is tested during the decode (second) phase of the pipeline; all other instructions (BCND, BCNDD, CC, CCD, RETC, and RETCD), test $\overline{\text{BIO}}$ during the execute (fourth) phase of the pipeline.

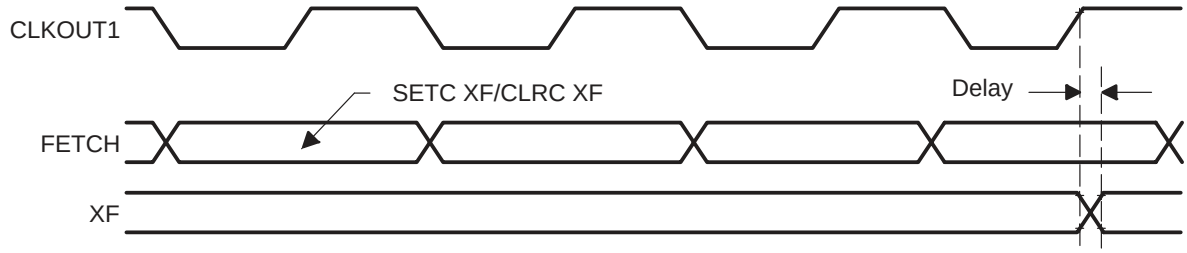
Figure 9–9. $\overline{\text{BIO}}$ Timing Diagram



9.5.2 External Flag Output (XF)

The XF pin signals to external devices via software. It is set high by the SETC XF instruction and reset low by the CLRC XF instruction. XF is set high at device reset. Figure 9–10 shows the relationship between the time the SETC or CLRC instruction is fetched, and the time the XF pin is set or reset (refer to the TMS320C5x data sheet for actual timing specifications). The timing diagram is for a sequence of single-cycle, single-word instructions located in external memory. Actual timing can vary with different instruction sequences.

Figure 9–10. XF Timing Diagram



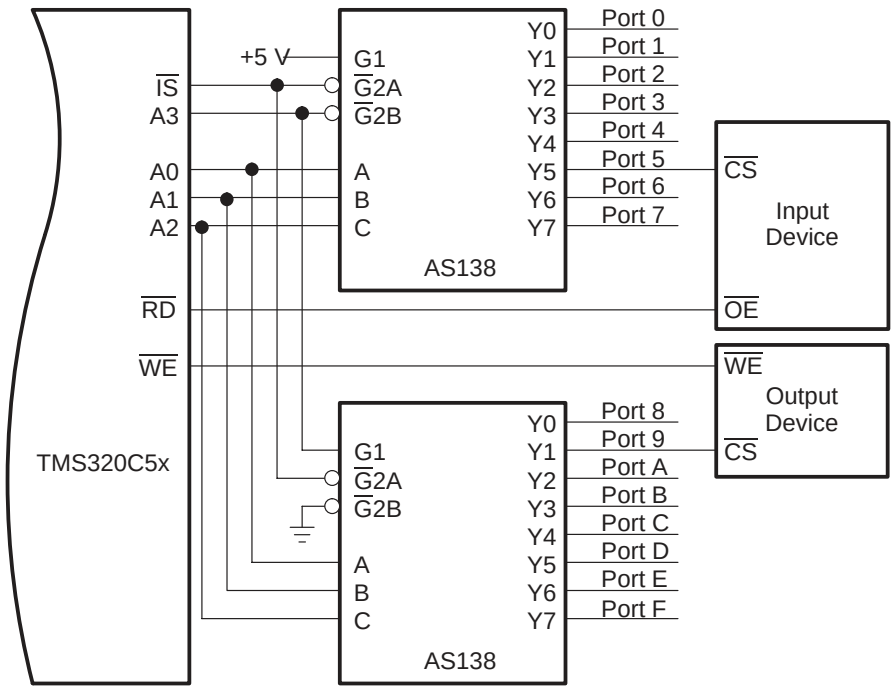
9.6 Parallel I/O Ports

The ’C5x has 64K parallel I/O ports. Sixteen of the 64K I/O ports are memory-mapped in data page 0 as listed in Table 9–1 on page 9-2. You can access the 64K I/O ports using the IN and OUT instructions or any instruction that reads or writes a location in data memory space. Accesses to memory-mapped I/O space are distinguished from program and data accesses by the $\overline{IS}$ signal going low; the $\overline{DS}$ signal is not active, even though the I/O port is actually accessed through data space. The following example shows how to use direct addressing to access an I/O device on port 51h:

```
SACL   51h    ;(DP = 0) Store accumulator to external
              ;device on port 81.
```

The $\overline{RD}$ signal can be used in conjunction with chip-select logic to generate an output enable signal for an external peripheral. The $\overline{WE}$ signal can be used in conjunction with chip-select logic to generate a write enable signal for an external peripheral. Figure 9–11 shows a typical I/O port interface circuitry. The decode section can be simplified if fewer I/O ports are used.

Figure 9–11. I/O Port Interface Circuitry



9.7 Serial Port Interface

Several ’C5x devices implement a variety of types of flexible serial port interfaces. These serial port interfaces provide full duplex, bidirectional, communication with serial devices such as codecs, serial analog to digital (A/D) converters, and other serial systems. The serial port interface signals are directly compatible with many industry-standard codecs and other serial devices. The serial port may also be used for interprocessor communication in multiprocessing applications (the time-division multiplexed (TDM) serial port is especially optimized for multiprocessing).

Three different types of serial port interfaces are available on ’C5x devices. The basic standard serial port interface (SP) is implemented on all ’C5x devices. The TDM serial port interface is implemented on the ’C50, ’C51, and ’C53 devices. The ’C56 and ’C57 devices include the buffered serial port (BSP), which implements an automatic buffering feature that greatly reduces CPU overhead required in handling serial data transfers. See Table 1–1 on page 1-6 for information about features included in various ’C5x devices.

The BSP operates in either autobuffering or nonbuffered mode. When operated in nonbuffered (or standard) mode, the BSP functions the same as the basic standard serial port (except where specifically indicated) and is described in this section. The TDM serial port operates in either TDM or non-TDM mode. When operated in non-TDM (or standard) mode, the TDM serial port also functions the same as the basic standard serial port and is described in this section.

The BSP also implements several enhanced features in standard mode, and these features, as well as operation of the BSP in autobuffering mode, are described in Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53. Therefore, when using the ’C56 or ’C57 devices, Section 9.8 should be consulted. Operation of the TDM serial port in TDM mode is described in Section 9.9, *Time-Division Multiplexed (TDM) Serial Port Interface*, on page 9-74. Note that the BSP and TDM serial ports initialize to a standard serial port compatible mode upon reset.

In all ’C5x serial ports, both receive and transmit operations are double-buffered, thus allowing a continuous communications stream with either 8- or 16-bit data packets. The continuous mode provides operation that, once initiated, requires no further frame synchronization pulses (FSR and FSX) when transmitting at maximum packet frequency. The serial ports are fully static and thus will function at arbitrarily low clocking frequencies. The maximum operating frequency for the standard serial port of one-fourth of CLKOUT1 (5M bps at 50 ns, 7.14M bps at 35 ns) is achieved when using internal serial port clocks. The maximum operating frequency for the BSP is CLKOUT1. When the serial

ports are in reset, the device may be configured to turn off the internal serial port clocks, allowing the device to run in a lower power mode of operation.

9.7.1 Serial Port Interface Registers

The serial port operates through the three memory-mapped registers (SPC, DXR, and DRR) and two other registers (RSR and XSR) that are not directly accessible to the program, but are used in the implementation of the double-buffering capability. These five registers are listed in Table 9–11.

Table 9–11. Serial Port Registers

Address	Register	Description
0020h	DRR	Data receive register
0021h	DXR	Data transmit register
0022h	SPC	Serial port control register
—	RSR	Receive shift register
—	XSR	Data transmit shift register

- Data receive register (DRR). The 16-bit memory-mapped data receive register (DRR) holds the incoming serial data from the RSR to be written to the data bus. At reset, the DRR is cleared.
- Data transmit register (DXR). The 16-bit memory-mapped data transmit register (DXR) holds the outgoing serial data from the data bus to be loaded in the XSR. At reset, the DXR is cleared.
- Serial port control register (SPC). The 16-bit memory-mapped serial port control register (SPC) contains the mode control and status bits of the serial port.
- Data receive shift register (RSR). The 16-bit data receive shift register (RSR) holds the incoming serial data from the serial data receive (DR) pin and controls the transfer of the data to the DRR.
- Data transmit shift register (XSR). The 16-bit data transmit shift register (XSR) controls the transfer of the outgoing data from the DXR and holds the data to be transmitted on the serial data transmit (DX) pin.

During normal serial port operation, the DXR is typically loaded with data to be transmitted on the serial port by the executing program, and its contents read automatically by the serial port logic to be sent out when a transmission is initiated. The DRR is loaded automatically by the serial port logic with data received on the serial port and read by the executing program to retrieve the received data.

At times during normal serial port operation, however, it may be desirable for a program to perform other operations with the memory-mapped serial port registers besides simply writing to DXR and reading from DRR.

On the SP, the DXR and DRR may be read or written at any time regardless of whether the serial port is in reset or not. On the BSP, access to these registers is restricted; the DRR can only be read, and the DXR can only be written when autobuffering is disabled (see subsection 9.8.2, *Autobuffering Unit (ABU) Operation*, on page 9-60). The DRR can only be written when the BSP is in reset. The DXR can be read at any time.

Note, however, that on both the SP and the BSP, care should be exercised when reading or writing to these registers during normal operation. With the DRR, since, as mentioned previously, this register is written automatically by the serial port logic when data is received, if a write to DRR is performed, subsequent reads may not yield the result written if a serial port receive occurs after the write but before the read is performed. With the DXR, care should be exercised when this register is written, since if previously written contents intended for transmission have not yet been sent, these contents will be overwritten and the original data lost. As mentioned previously, the DXR can be read at any time.

Alternatively, DXR and DRR may also serve as general purpose storage if they are not required for serial port use. If these registers are to be used for general purpose storage, the transmit and/or receive sections of the serial port should be disabled either by tying off (by pulling up or down, whichever is appropriate) external input pins which could spuriously cause serial port transfers, or by putting the port in reset.

9.7.2 Serial Port Interface Operation

This section describes operation of the basic standard serial port interface, which includes operation of the TDM and BSP serial ports when configured in standard mode. Table 9–12 lists the pins used in serial port operation. Figure 9–12 shows these pins for two ’C5x serial ports connected for a one-way transfer from device 0 to device 1. Only three signals are required to connect from a serial port transmitter to a receiver for data transmission. The transmitted serial data signal (DX) sends the actual data. The transmit frame synchronization signal (FSX) initiates the transfer (at the beginning of the packet), and the transmit clock signal (CLKX) clocks the bit transfer. The corresponding pins on the receive device are DR, FSR and CLKR, respectively.

Table 9–12. Serial Port Pins

Pin	Description
CLKR	Receive clock signal
CLKX	Transmit clock signal
DR	Received serial data signal
DX	Transmitted serial data signal
FSR	Receive framing synchronization signal
FSX	Transmit frame synchronization signal

Figure 9–12. One-Way Serial Port Transfer

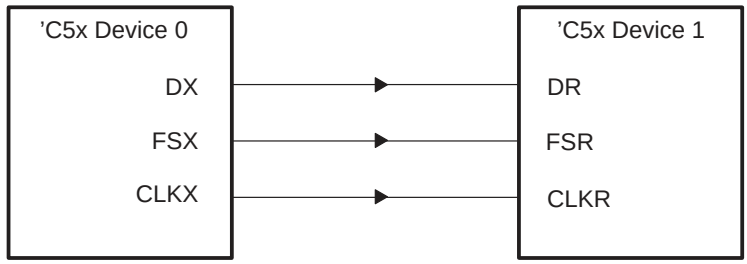
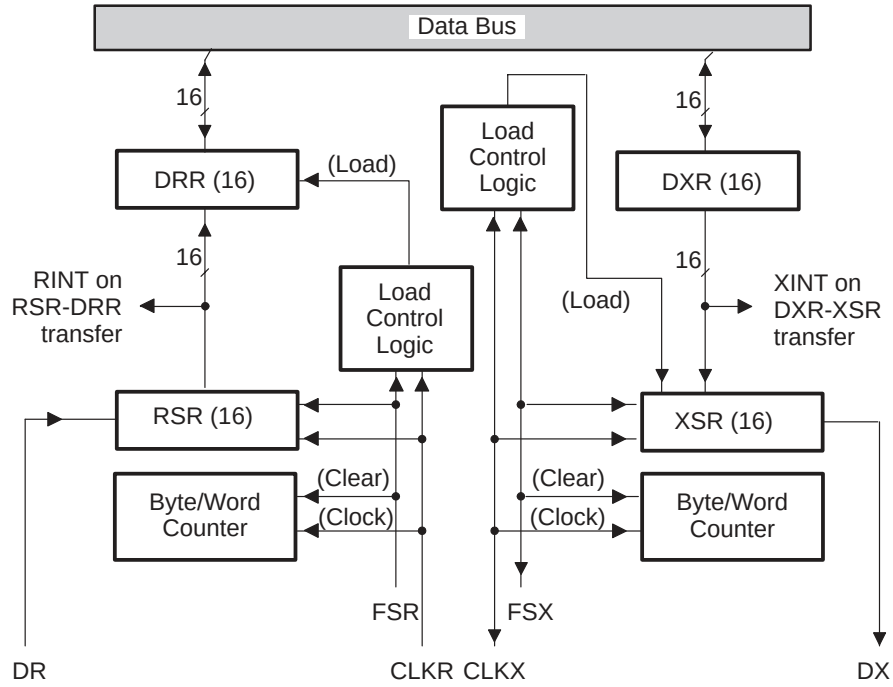


Figure 9–13 shows how the pins and registers are configured in the serial port logic and how the double-buffering is implemented.

Transmit data is written to the DXR, while received data is read from the DRR. A transmit is initiated by writing data to the DXR, which copies the data to the XSR when the XSR is empty (when the last word has been transmitted serially, that is, driven on the DX pin). The XSR manages shifting the data to the DX pin, thus allowing another write to DXR as soon as the DXR-to-XSR copy is completed.

During transmits, upon completion of the DXR-to-XSR copy, a 0-to-1 transition occurs on the transmit ready (XRDY) bit in the SPC. This 0-to-1 transition generates a serial port transmit interrupt (XINT) that signals that the DXR is ready to be reloaded. See Section 4.8, *Interrupts*, on page 4-36 and subsection 9.1.2, *External Interrupts*, on page 9-4 for more information on 'C5x interrupts.

Figure 9–13. Serial Port Interface Block Diagram

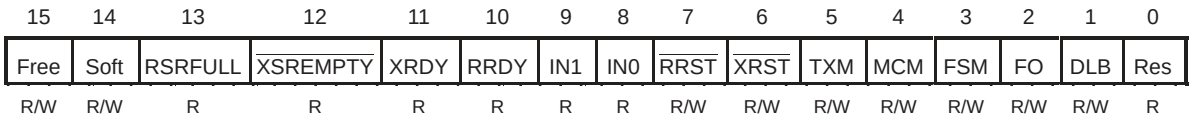


The process is similar in the receiver. Data from the DR pin is shifted into the RSR, which is then copied into the DRR from which it may be read. Upon completion of the RSR-to-DRR copy, a 0-to-1 transition occurs on the receive ready (RRDY) bit in the SPC. This 0-to-1 transition generates a serial port receive interrupt (RINT). Thus, the serial port is double-buffered because data can be transferred to or from DXR or DRR while another transmit or receive is being performed. Note that transfer timing is synchronized by the frame sync pulse in burst mode (discussed in more detail in subsection 9.7.4, *Burst Mode Transmit and Receive Operations*, on page 9-37).

9.7.3 Setting the Serial Port Configuration

The SPC contains control bits which configure the operation of the serial port. The SPC bit fields are shown in Figure 9–14 and described in Table 9–13. Note that seven bits in the SPC are read only and the remaining nine bits are read/write.

Figure 9–14. Serial Port Control Register (SPC) Diagram



Note: R = Read, W = Write

Table 9–13. Serial Port Control Register (SPC) Bit Summary

Bit	Name	Reset Value	Function
15	Free	0	This bit is used in conjunction with the Soft bit to determine the state of the serial port clock when a halt is encountered. See Table 9–14 on page 9-37 for the serial port clock configurations. Free = 0 — The Soft bit selects the emulation mode. Free = 1 — The serial port clock runs free regardless of the Soft bit.
14	Soft	0	This bit is used in conjunction with the Free bit to determine the state of the serial port clock when a halt is encountered. When the Free bit is cleared to 0, the Soft bit selects the emulation mode. See Table 9–14 on page 9-37 for the serial port clock configurations. Soft = 0 — The serial port clock stops immediately, thus aborting any transmission. Soft = 1 — The clock stops after completion of the current transmission.
13	RSRFULL	0	Receive Shift Register Full. This bit indicates whether the receiver has experienced overrun. Overrun occurs when RSR is full and DRR has not been read since the last RSR-to-DRR transfer. On the SP, when FSM = 1, the occurrence of a frame sync pulse on FSR qualifies the generation of RSRFULL = 1. When FSM = 0, and on the BSP, only the basic two conditions apply; that is, RSRFULL goes high without waiting for an FSR pulse. RSRFULL = 0 — Any one of the following three events clears the RSRFULL bit to 0: reading DRR, resetting the receiver ($\overline{\text{RRST}}$ bit to 0), or resetting the device. RSRFULL = 1 — The port has recognized an overrun. When RSRFULL = 1, the receiver halts and waits for DRR to be read, and any data sent on DR is lost. On the SP, the data in RSR is preserved; on the BSP, the contents of RSR are lost.

Table 9–13. Serial Port Control Register (SPC) Bit Summary (Continued)

Bit	Name	Reset Value	Function
12	$\overline{\text{XSREMPTY}}$	0	Transmit Shift Register Empty. This bit indicates whether the transmitter has experienced underflow. Underflow occurs when XSR is empty and DXR has not been loaded since the last DXR-to-XSR transfer. $\overline{\text{XSREMPTY}}$ = 0 — Any one of the following three events clears the $\overline{\text{XSREMPTY}}$ bit to 0: underflow has occurred, resetting the transmitter ($\overline{\text{XRST}}$ bit to 0), or resetting the device. $\overline{\text{XSREMPTY}}$ = 1 — On the SP, $\overline{\text{XSREMPTY}}$ is deactivated (set to 1) directly as a result of writing to DXR; on the BSP, $\overline{\text{XSREMPTY}}$ is only deactivated after DXR is loaded *followed* by the occurrence of an FSX pulse.
11	XRDY	1	Transmit Ready. A transition from 0 to 1 of the XRDY bit indicates that the DXR contents have been copied to XSR and that DXR is ready to be loaded with a new data word. A transmit interrupt (XINT) is generated upon the transition. This bit can be polled in software instead of using serial port interrupts. Note that on the SP, XRDY is generated directly as a result of writing to DXR; while on the BSP, XRDY is only generated after DXR is loaded *followed* by the occurrence of an FSX pulse. At reset or serial port transmitter reset ($\overline{\text{XRST}}$ = 0), the XRDY bit is set to 1.
10	RRDY	0	Receive Ready. A transition from 0 to 1 of the RRDY bit indicates that the RSR contents have been copied to the DRR and that the data can be read. A receive interrupt (RINT) is generated upon the transition. This bit can be polled in software instead of using serial port interrupts. At reset or serial port receiver reset ($\overline{\text{RRST}}$ = 0), the RRDY bit is cleared to 0.
9	IN1	x	Input 1. This bit allows the CLKX pin to be used as a bit input. IN1 reflects the current level of the CLKX pin of the device. When CLKX switches levels, there is a latency of between 0.5 and 1.5 CLKOUT1 cycles before the new CLKX value is represented in the SPC.
8	IN0	x	Input 0. This bit allows the CLKR pin to be used as a bit input. IN0 reflects the current level of the CLKR pin of the device. When CLKR switches levels, there is a latency of between 0.5 and 1.5 CLKOUT1 cycles before the new CLKR value is represented in the SPC.
7	$\overline{\text{RRST}}$	0	Receive Reset. This signal resets and enables the receiver. When a 0 is written to the $\overline{\text{RRST}}$ bit, activity in the receiver halts. $\overline{\text{RRST}}$ = 0 — The serial port receiver is reset. Writing a 0 to $\overline{\text{RRST}}$ clears the RSRFULL and RRDY bits to 0. $\overline{\text{RRST}}$ = 1 — The serial port receiver is enabled.

Table 9–13. Serial Port Control Register (SPC) Bit Summary (Continued)

Bit	Name	Reset Value	Function
6	$\overline{\text{XRST}}$	0	Transmitter Reset. This signal is used to reset and enable the transmitter. When a 0 is written to the $\overline{\text{XRST}}$ bit, activity in the transmitter halts. When the XRDY bit is 0, writing a 0 to $\overline{\text{XRST}}$ generates a transmit interrupt (XINT). $\overline{\text{XRST}}$ = 0 — The serial port transmitter is reset. Writing a 0 to $\overline{\text{XRST}}$ clears the $\overline{\text{XSREMPTY}}$ bit to 0 and sets the XRDY bit to 1. $\overline{\text{XRST}}$ = 1 — The serial port transmitter is enabled.
5	TXM	0	Transmit Mode. This bit configures the FSX pin as an input (TXM = 0) or as an output (TXM = 1). TXM = 0 — *External frame sync*. The transmitter idles until a frame sync pulse is supplied on the FSX pin. TXM = 1 — *Internal frame sync*. Frame sync pulses are generated internally when data is transferred from the DXR to XSR to initiate data transfers. The internally generated framing signal is synchronous with respect to CLKX.
4	MCM	0	Clock Mode. This bit specifies the clock source for CLKX. MCM = 0 — CLKX is taken from the CLKX pin. MCM = 1 — CLKX is driven by an on-chip clock source. For the SP and the BSP in standard mode, this on-chip clock source is at a frequency of one-fourth of CLKOUT1. The BSP also allows the option of generating clock frequencies at additional ratios of CLKOUT1. For a detailed description of this feature, see Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53. Note that if MCM = 1 and DLB = 1, a CLKR signal is also supplied by the internal source.
3	FSM	0	Frame Sync Mode. This bit specifies whether frame synchronization pulses (FSX and FSR) are required after the initial frame sync pulse for serial port operation. See subsection 9.7.2, *Serial Port Interface Operation*, on page 9-25 for more details on the frame sync signals. FSM = 0 — *Continuous mode*. Frame sync pulses are not required after the initial frame sync pulse, but they are not ignored; therefore, improperly timed frame syncs may cause errors in serial transfers. See subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46 for information about serial port operation under various exception conditions. FSM = 1 — *Burst mode*. A frame sync pulse is required on FSX/FSR for the transmission/reception of each word.

Table 9–13. Serial Port Control Register (SPC) Bit Summary (Continued)

Bit	Name	Reset Value	Function
2	FO	0	Format. This bit specifies the word length of the serial port transmitter and receiver.
			FO = 0 — The data is transmitted and/or received as 16-bit words.
			FO = 1 — The data is transferred as 8-bit bytes. The data is transferred with the MSB first. The BSP also allows the capability of 10- and 12-bit transfers. For a detailed description of this feature, see Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53.
1	DLB	0	Digital Loopback Mode. This bit can be used to put the serial port in digital loopback mode.
			DLB = 0 — The digital loopback mode is disabled. The DR, FSR, and CLKR signals are taken from their respective device pins.
			DLB = 1 — The digital loopback mode is enabled. The DR and FSR signals are connected to DX and FSX, respectively, through multiplexers, as shown in Figure 9–15(a) and (b) on page 9-32. Additionally, CLKR is driven by CLKX if MCM = 1. If DLB = 1 and MCM = 0, CLKR is taken from the CLKR pin of the device. This configuration allows CLKX and CLKR to be tied together externally and supplied by a common external clock source. The logic diagram for CLKR is shown in Figure 9–15(c) on page 9-32. Note also that in DLB mode, the FSX and DX signals appear on the device pins, but FSR and DR do not. Either internal or external FSX signals may be used in DLB mode, as defined by the TXM bit.
0	Res	0	Reserved. Always read as a 0 in the serial port. This bit performs a function in the TDM serial port discussed in Section 9.9, *Time-Division-Multiplexed (TDM) Serial Port Interface*, on page 9-74.

Reserved Bit

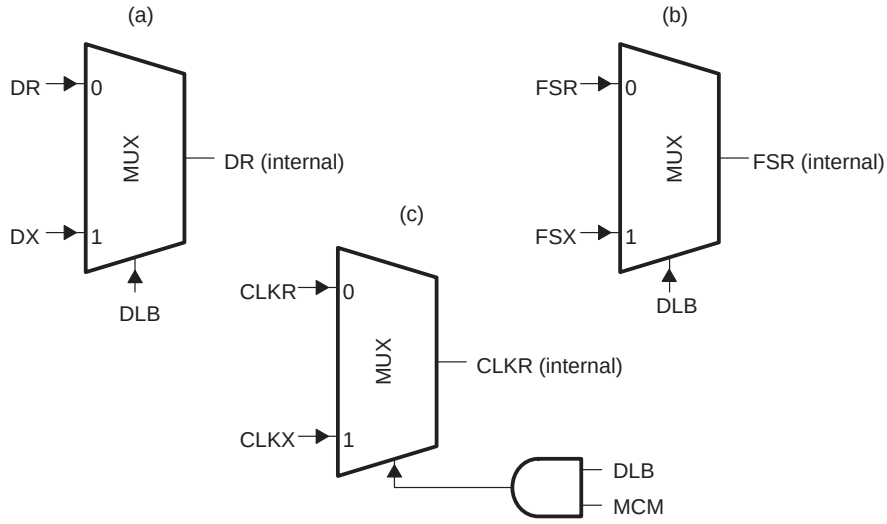
Bit 0 is reserved and is read as 0, although it performs a function in the TDM serial port (discussed in Section 9.9, *Time-Division-Multiplexed (TDM) Serial Port Interface*, on page 9-74).

DLB Bit

The DLB (bit 1) selects digital loopback mode, which allows testing of serial port code with a single ’C5x device. When DLB = 1, DR and FSR are connected to DX and FSX, respectively, through multiplexers, as shown in Figure 9–15.

When in loopback mode, CLKR is driven by CLKX if on-chip serial port clock generation is selected (MCM = 1), but if MCM = 0, then CLKR is driven by the external CLKR signal. This allows for the capability of external serial port clock generation in digital loopback mode. If DLB = 0, then normal operation occurs where DR, FSR, and CLKR are all taken from their respective pins.

Figure 9–15. Receiver Signal MUXes



FO Bit

The FO (bit 2) specifies whether data is transmitted as 16-bit words (FO = 0) or 8-bit bytes (FO = 1). Note that in the latter case, only the lower byte of whatever is written to DXR is transmitted, and the lower byte of data read from DRR is what was received. To transmit a whole 16-bit word in 8-bit mode, two writes to DXR are necessary, with the appropriate shifts of the value because the upper eight bits written to DXR are ignored. Similarly, to receive a whole 16-bit word in 8-bit mode, two reads from DRR are required, with the appropriate shifts of the value. In the SP, the upper eight bits of DRR are indeterminate in 8-bit receptions; in the BSP, the unused bits of DRR are sign-extended. Additionally, in the BSP, transfers of 10- and 12-bit words are provided for additional flexibility. For a detailed description of this feature, refer to Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53.

FSM Bit

The FSM (bit 3) specifies whether or not frame sync pulses are required in consecutive serial port transmits. If FSM = 1, a frame sync must be present for every transfer, although FSX may be either externally or internally generated depending on TXM. This mode is referred to as burst mode, because there are normally periods of inactivity on the serial port between transmits.

The frequency with which serial port transmissions occur is called packet frequency, and data packets can be 8, 10, 12, or 16 bits long. Therefore, as packet frequency increases, it reaches a maximum that occurs when the time, in serial port clock cycles, from one packet to the next, is equal to the number of bits being transferred. If transmission occurs at the maximum rate for multiple transfers in a row, however, frame sync essentially becomes redundant. Note that frame sync actually becomes redundant in burst mode only at maximum packet frequency with FSX configured as an output (TXM = 1). When FSX is an input (TXM = 0), its presence is required for transmissions to occur.

FSM = 0 selects the continuous mode of operation which requires only an initial frame sync pulse as long as a write to DXR (for transmit), or a read from DRR (for receive), is executed during each transfer. Note that when FSM = 0, frame sync pulses are not required, but they are not ignored, therefore, improperly timed frame syncs may cause errors in serial transfers. The timing of burst and continuous modes is discussed in detail in subsections 9.7.4, 9.7.5, and 9.7.6.

MCM Bit

The serial port clock source is set by MCM (bit 4). If MCM = 0, CLKX is configured as an input and thus accepts an external clock. If MCM = 1, then CLKX is configured as an output, and is driven by an internal clock source. For the SP, and the BSP operating in standard mode, this on-chip clock is at a frequency of one-fourth of CLKOUT1. The BSP also allows the option of generating clock frequencies at additional ratios of CLKOUT1. For a detailed description of this feature, refer to Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53. Note that the CLKR pin is always configured as an input.

TXM Bit

The transmit frame synchronization pulse source is set by TXM (bit 5). Like MCM, if TXM = 1, FSX is configured as an output and generates a pulse at the beginning of every transmit. If TXM = 0, FSX is configured as an input, and accepts an external frame sync signal. Note that the FSR pin is always configured as an input.

$\overline{\text{XRST}}$ and $\overline{\text{RRST}}$ Bits

The serial port transmitter and receiver are reset with $\overline{\text{XRST}}$ (bit 6) and $\overline{\text{RRST}}$ (bit 7). These signals are active low, so that if $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 0, the serial port is in a reset state. To reset and reconfigure the serial port, a total of two writes to the SPC are required.

- The first write to the SPC should:
  - write a 0 to the $\overline{\text{XRST}}$ and $\overline{\text{RRST}}$ bits
  - write the desired configuration to the remainder of the bits.

- The second write to the SPC should:
  - write 1 to the $\overline{\text{XRST}}$ and $\overline{\text{RRST}}$ bits
  - resend the desired configuration to the remainder of the bits.

The second write takes the serial port out of reset. Note that the transmitter and receiver may be reset individually if desired. When a 0 is written to $\overline{\text{XRST}}$ or $\overline{\text{RRST}}$, activity in the corresponding section of the serial port stops. This minimizes the switching and allows the device to operate with lower power consumption. When $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = MCM = 0, power requirements are further reduced since CLKX is no longer driven as an output.

Note that in IDLE2 mode, SP operation halts as with other parts of the ’C5x device. On the BSP, however, if the external serial port clock is being used, operation continues after an IDLE2 is executed. This allows power savings to still be realized in IDLE2 mode, while still maintaining operation of critical serial port functions if necessary (see Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53 for further information about BSP operation).

It should also be noted that, on the SP, the serial port may be taken out of reset at any time. Depending on the timing of exiting reset, however, a frame sync pulse may be missed. On the BSP, for receive and transmit with external frame sync, a setup of at least one CLKOUT1 cycle plus 1/2 serial port clock cycle is required prior to FSX being sampled active in standard mode. In autobuffering mode, additional setup is required (see Section 9.8, *Buffered Serial Port (BSP) Interface*, on page 9-53 for further information about BSP initialization timing requirements).

IN0 and IN1 Bits

IN0 (bit 8) and IN1 (bit 9) allow the CLKR and CLKX pins to be used as bit inputs. IN0 and IN1 reflect the current states of the CLKR and CLKX pins. The data on these pins can be sampled by reading the SPC. This can be accomplished using the BIT instruction (page 6-63), BITT instruction (page 6-65), or PLU instructions (Table 6–6 on page 6-14). Note that there is a latency of between 0.5 and 1.5 CLKOUT1 cycles in duration from CLKR/CLKX switching to the new CLKR/CLKX value being available in the SPC. Note that even if the serial port is reset, IN0 and IN1 can still be used as bit inputs, and DRR and DXR as general-purpose registers.

RRDY and XRDY Bits

Bits 10–13 in the SPC are read-only status bits that indicate various states of serial port operation. Writes and reads of the serial port may be synchronized by polling RRDY (bit 10) and XRDY (bit 11), or by using the interrupts that they generate. A transition from 0 to 1 of the RRDY bit indicates that the RSR contents have been copied to the DRR and that the received data may be read. A receive interrupt (RINT) is generated upon this transition.

A transition from 0 to 1 of the XRDY bit indicates that the DXR contents have been copied to XSR and that DXR is ready to be loaded with a new data word. A transmit interrupt (XINT) is generated upon this transition. Polling XRDY and RRDY in software may either substitute for or complement the use of serial port interrupts (both polling and interrupts may be used together if so desired). Note that with external FSX, on the SP, XSR is loaded directly as a result of loading DXR, while on the BSP, XSR is not loaded until an FSX occurs.

$\overline{\text{XSREMPTY}}$ Bit

The $\overline{\text{XSREMPTY}}$ (bit 12) indicates whether the transmitter has experienced underflow. $\overline{\text{XSREMPTY}}$ is an active low bit; therefore, when $\overline{\text{XSREMPTY}}$ = 0, an underflow has occurred.

Any *one* of the following three conditions causes $\overline{\text{XSREMPTY}}$ to become active ($\overline{\text{XSREMPTY}}$ = 0):

- DXR has not been loaded since the last DXR-to-XSR transfer, and XSR empties (the actual transition of $\overline{\text{XSREMPTY}}$ occurs after the last bit has been shifted out of XSR),
- or the transmitter is reset ($\overline{\text{XRST}}$ = 0),
- or the ’C5x device is reset ($\overline{\text{RS}}$ = 0).

When $\overline{\text{XSREMPTY}}$ = 0, the transmitter halts and stops driving DX (the DX pin is in a high-impedance state) until the next frame sync pulse. Note that underflow does not constitute an error condition in the burst mode, although it does in the continuous mode (error conditions are further discussed in subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46).

The following condition causes $\overline{\text{XSREMPTY}}$ to become inactive ($\overline{\text{XSREMPTY}}$ = 1):

- A write to DXR occurs on the SP, or on the BSP a write to DXR occurs followed by an FSX pulse (see subsection 9.7.4, *Burst Mode Transmit and Receive Operations*, on page 9-37 for further information about transmit timing).

RSRFULL Bit

The RSRFULL (bit 13) indicates whether the receiver has experienced overrun. RSRFULL is an active high bit; therefore, when RSRFULL = 1, RSR is full.

In burst mode (FSM = 1), all three of the following must occur to cause RSRFULL to become active (RSRFULL = 1):

- The DRR has not been read since the last RSR-to-DRR transfer,
- RSR is full,
- and a frame sync pulse appears on FSR.

In continuous mode (FSM = 0), and on the BSP, only the first two conditions are necessary to set RSRFULL:

- The DRR has not been read since the last RSR-to-DRR transfer
- and RSR is full.

Therefore, in continuous mode, and on the BSP, RSRFULL occurs after the last bit has been received.

When RSRFULL = 1, the receiver halts and waits for the DRR to be read, and any data sent on DR is lost. On the SP, the data in RSR is preserved; on the BSP, the RSR contents are lost.

Any *one* of the following three conditions causes RSRFULL to become inactive (RSRFULL = 0):

- The DRR is read,
- or the serial port is reset ($\overline{\text{RRST}}$ = 0),
- or the ’C5x device is reset ($\overline{\text{RS}}$ = 0).

Soft and Free Bits

Soft (bit 14) and Free (bit 15) are special emulation bits that determine the state of the serial port clock when a breakpoint is encountered in the high-level language debugger. If the Free bit is set to 1, then upon a software breakpoint, the clock continues to run (free runs) and data is still shifted out. When Free = 1, the Soft bit is a *don't care*. If the Free bit is cleared to 0, then the Soft bit takes effect. If the Soft bit is cleared to 0, then the clock stops immediately, thus aborting any transmission. If the Soft bit is set to 1 and a transmission is in progress, the transmission continues until completion of the transfer, and then the clock halts. These options are listed in Table 9–14.

The receive side functions in a similar fashion. Note that if an option other than *immediate stop* (Soft = Free = 0) is chosen, the receiver continues running and an overflow error is possible. The default value for these bits is *immediate stop*.

Table 9–14. Serial Port Clock Configuration

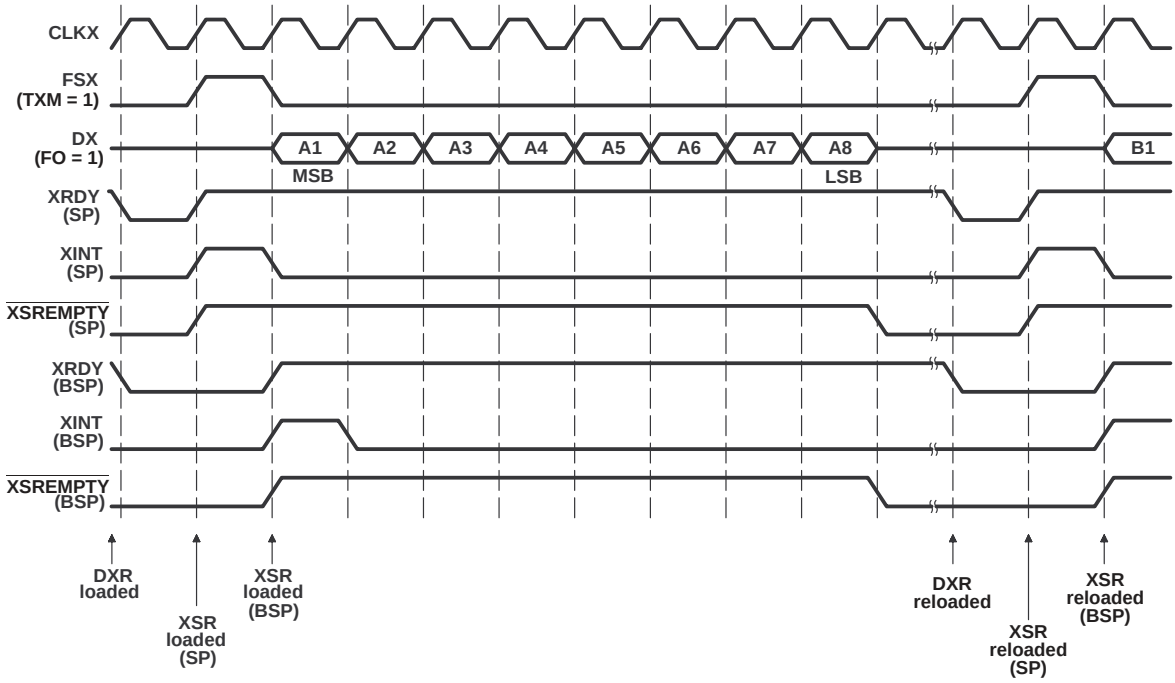
Free	Soft	Serial Port Clock Configuration
0	0	Immediate stop, clocks are stopped. (Reset values)
0	1	Transmitter stops after completion of word. Receiver is not affected.
1	X	Free run.

Note: X = Don't care

9.7.4 Burst Mode Transmit and Receive Operations

In burst mode operation, there are periods of serial port inactivity between packet transmits. The data packet is marked by the frame sync pulse occurring on FSX (see Figure 9–16). On the transmit device, the transfer is initiated by a write to DXR. The value in DXR is then transferred to XSR, and, upon a frame sync pulse on FSX (generated internally or externally depending on TXM), the value in XSR is shifted out and driven on the DX pin. Note that on the SP, the DXR to XSR transfer occurs on the second rising edge of CLKX after DXR is loaded, while on the BSP this transfer does not occur until an FSX occurs, when FSX is external. When FSX is internal on the BSP, the DXR to XSR transfer and generation of FSX occur directly after loading DXR. On both the SP and the BSP, once XSR is loaded with the value from DXR, XRDY goes high, generating a transmit interrupt (XINT) and setting $\overline{\text{XSREMPTY}}$ to a 1.

Figure 9–16. Burst Mode Serial Port Transmit Operation



Note that in both the SP and the BSP, DXR to XSR transfers occur only if the XSR is empty and the DXR has been loaded since the last DXR to XSR transfer. If DXR is reloaded before the old DXR contents have been transferred to XSR, the previous DXR contents are overwritten. Accordingly, unless overwriting DXR is intended, the DXR should only be loaded if XRDY = 1. This is assured if DXR writes are made only in response to a transmit interrupt or polling XRDY.

It should be noted that in the following discussions, the timings are slightly different for internally (TXM = 1, FSX is an output) and externally (TXM = 0, FSX is an input) generated frame syncs. This distinction is made because in the former case, the frame sync pulse is generated by the transmitting device as a direct result of a write to DXR. In the latter case, there is no such direct effect. Instead, the transmitting device must write to DXR and wait for an externally generated frame sync.

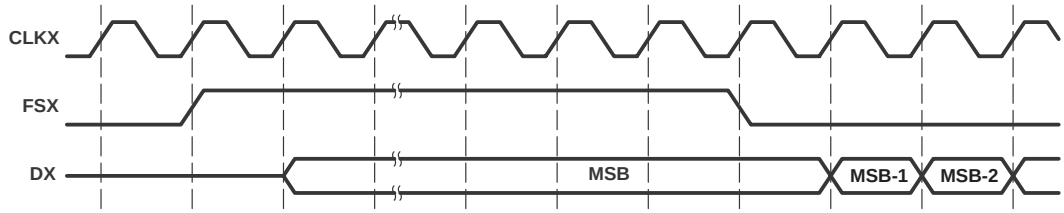
If internal frame sync pulse generation is selected (TXM = 1), a frame sync pulse is generated on the second rising edge of CLKX following a write to DXR. For externally generated frame syncs, the events described here will occur as soon as a properly timed frame sync pulse occurs (see the data sheet for detailed serial port interface timings).

On the next rising edge of CLKX after FSX goes high, the first data bit (MSB first) is driven on the DX pin. Thus, if the frame sync pulse is generated internally (TXM = 1), there is a 2-CLKX cycle latency (approximately) after DXR is loaded, before the data is driven on the line. If frame sync is externally generated, data transmission is delayed indefinitely after a DXR load until the FSX pulse occurs (this is described in further detail later in this subsection). With the falling edge of frame sync, the rest of the bits are shifted out. When all the bits are transferred, DX enters a high-impedance state.

At the end of each transmission, if DXR was not reloaded when XINT was generated, $\overline{\text{XSREMPTY}}$ becomes active (low) at this point, indicating underflow. With externally generated frame sync, if $\overline{\text{XSREMPTY}}$ is active and a frame sync pulse is generated, any old data in the DXR is transmitted. This is explained in detail in subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46.

Note that the first data bit transferred could have variable length if frame sync is generated externally and does not fall within one CLKX cycle (this is illustrated in Figure 9–17). Internally generated frame syncs are assured by ’C5x timings to be one CLKX cycle in duration.

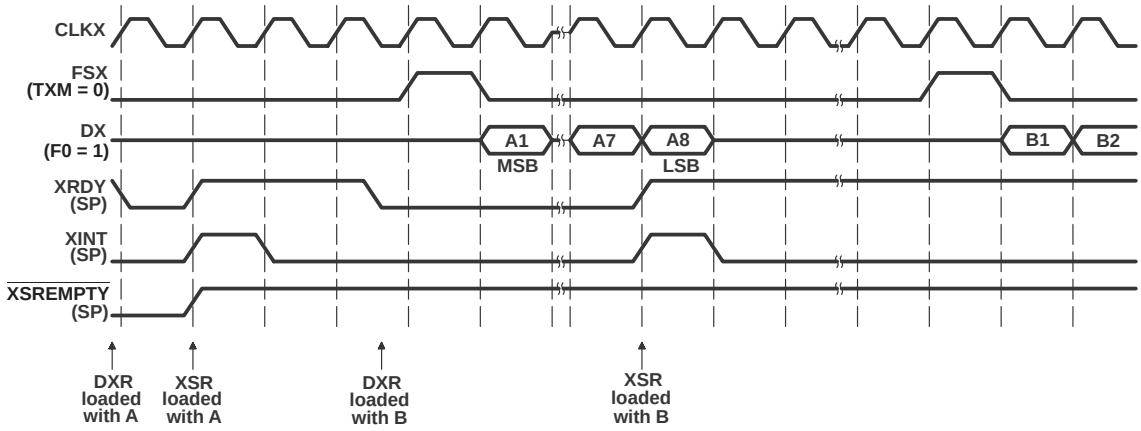
Figure 9–17. Serial Port Transmit With Long FSX Pulse



Serial port transmit with external frame sync pulses is similar to that with internal frame sync, with the exception that transfers do not actually begin until the external frame sync occurs. If the external frame sync occurs many CLKX cycles after DXR is loaded, however, the double buffer is filled and frozen until frame sync appears.

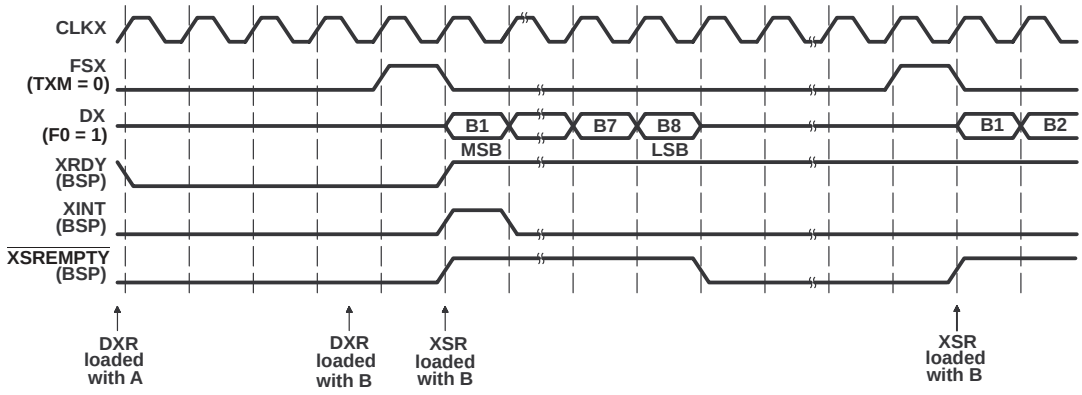
On the SP (Figure 9–18), when the delayed frame sync occurs, A is transmitted on DX; after the transmit, a DXR-to-XSR copy of B occurs, XINT is generated, and again, the transmitter remains frozen until the next frame sync. When frame sync finally occurs, B is transmitted on DX. Note that when B is loaded into DXR, a DXR-to-XSR copy of B does not occur immediately because A has not been transmitted, and no XINT is generated. Any subsequent writes to DXR before the next delayed frame sync occurs overwrite B in the DXR.

Figure 9–18. Burst Mode Serial Port Transmit Operation With Delayed Frame Sync in External Frame Sync Mode (SP)



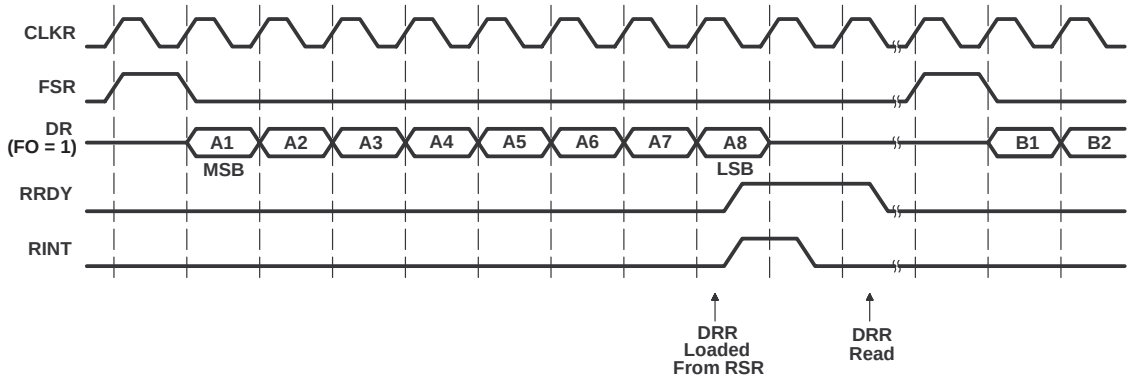
On the BSP (Figure 9–19), since DXR was reloaded with B shortly after being loaded with A when the delayed frame sync finally occurs, B is transmitted on DX. After the transmit, the transmitter remains frozen until the next frame sync. When frame sync finally occurs, B is again transmitted on DX. Note that when B is loaded into DXR, a DXR-to-XSR copy of B does not occur immediately since the BSP requires a frame sync to initiate transmitting. Any subsequent writes to DXR before the next delayed frame sync occurs overwrite B in the DXR.

Figure 9–19. Burst Mode Serial Port Transmit Operation With Delayed Frame Sync in External Frame Sync Mode (BSP)



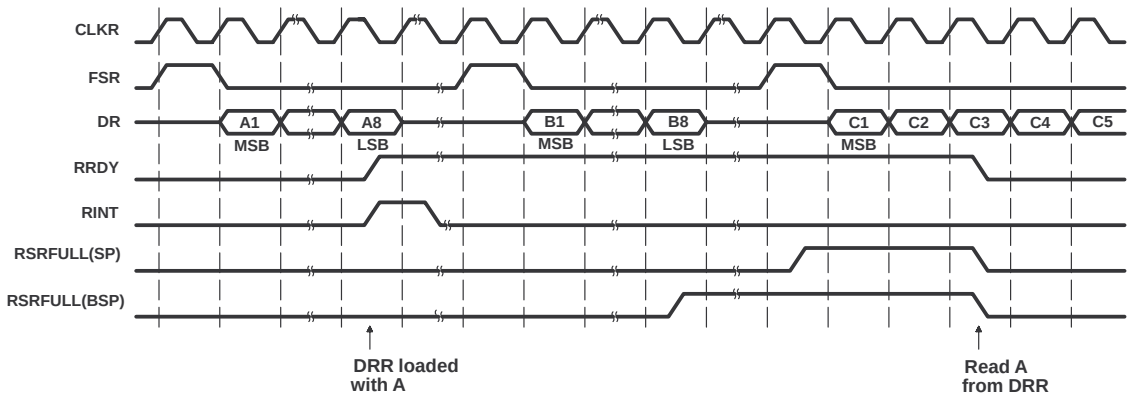
During a receive operation, shifting into RSR begins on the falling edge of the CLKR cycle after frame sync has gone low (as shown in Figure 9–20). Then, as the last data bit is being received, the contents of the RSR are transferred to the DRR on the falling edge of CLKR, and RRDY goes high, generating a receive interrupt (RINT).

Figure 9–20. Burst Mode Serial Port Receive Operation



If the DRR from a previous receive has not been read, and another word is received, no more bits can be accepted without causing data corruption since DRR and RSR are both full. In this case, the RSRFULL bit is set indicating this condition. On the SP, this occurs with the next FSR; on the BSP, RSRFULL is set on the falling edge of CLKR during the last bit received. RSRFULL timing on both the SP and BSP is shown in Figure 9–21.

Figure 9–21. Burst Mode Serial Port Receive Overrun



Unlike transmit underflow, overrun (RSRFULL = 1) constitutes an actual error condition. While DRR contents are preserved in overrun, its occurrence can often result in loss of other received data.

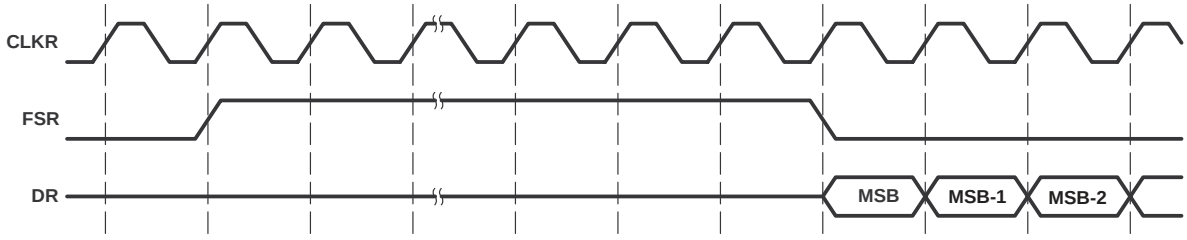
Overrun is handled differently on the SP and on the BSP. On the SP, the contents of RSR are preserved on overrun, but since RSRFULL is not set to 1 until the next FSR occurs after the overflowing reception, incoming data usually begins being lost as soon as RSRFULL is set. Data loss can only be avoided if RSRFULL is polled in software and the DRR is read immediately after RSRFULL is set to 1. This is normally possible only if the CLKR frequency is slow with respect to CLKOUT1, since RSRFULL is set on the falling edge of CLKR during FSR, and data begins being received on the following rising edge of CLKR. The time available for polling RSRFULL and reading the DRR to avoid data loss is, therefore, only half of one CLKR cycle.

On the BSP, RSRFULL is set on the last valid bit received, but the contents of RSR are never transferred to DRR, therefore, the complete transferred word in RSR is lost. If the DRR is read (clearing RSRFULL) before the next FSR occurs, subsequent transfers can be received properly.

Overrun and various other serial port exception conditions such as the occurrence of frame sync during a receive are discussed in further detail in subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46.

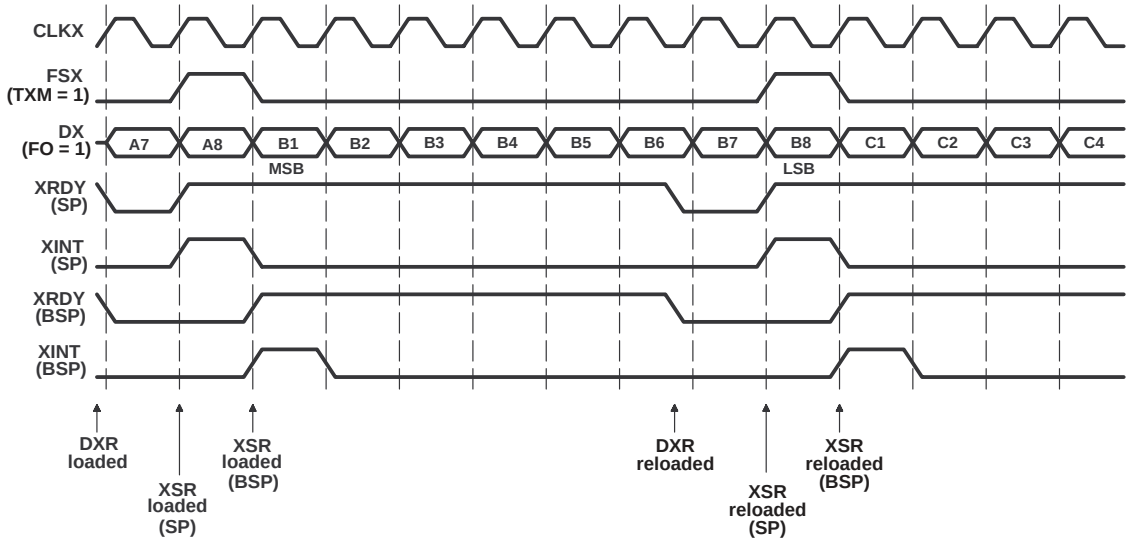
If the serial port receiver is provided with FSR pulses significantly longer than one CLKR cycle, timing of data reception is effected in a similar fashion as with long FSX pulses. With long FSR pulses, however, the reception of all bits, including the first one, is simply delayed until FSR goes low. Serial port receive operation with a long FSR pulse is illustrated in Figure 9–22.

Figure 9–22. Serial Port Receive With Long FSR pulse



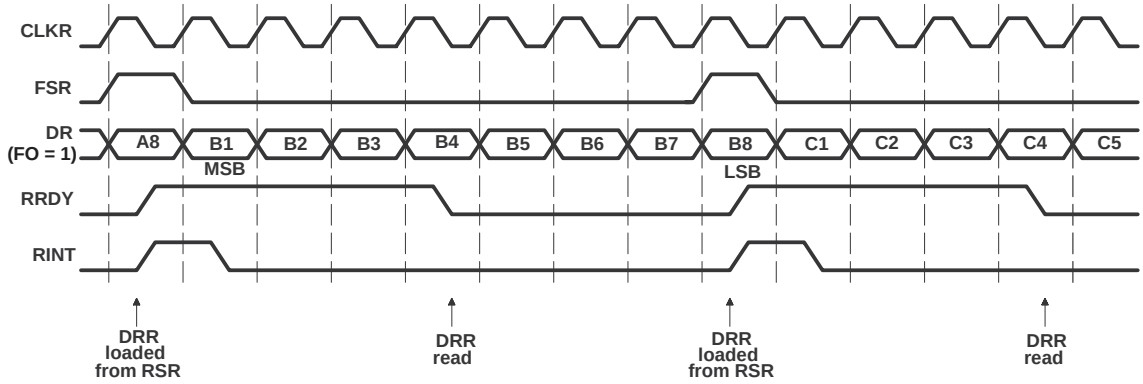
Note that if the packet transmit frequency is increased, the inactivity period between the data packets for adjacent transfers decreases to zero. This corresponds to a minimum period between frame sync pulses (equivalent to 8 or 16 CLKX/R cycles, depending on FO) that corresponds to a maximum packet frequency at which the serial port may operate. At maximum packet frequency, transmit timing is a compressed version of Figure 9–16, as shown in Figure 9–23.

Figure 9–23. Burst Mode Serial Port Transmit at Maximum Packet Frequency



At maximum packet frequency, the data bits in consecutive packets are transmitted contiguously with no inactivity between bits. The frame sync pulse overlaps the last bit transmitted in the previous packet. Maximum packet frequency receive timing is similar and is shown in Figure 9–24.

Figure 9–24. Burst Mode Serial Port Receive at Maximum Packet-Frequency



As shown in Figure 9–23 and Figure 9–24, with the transfer of multiple data packets at maximum packet frequency in burst mode, packets are transmitted at a constant rate, and the serial port clock provides sufficient timing information for the transfer, which permits a continuous stream of data. Therefore, the frame sync pulses are essentially redundant. Theoretically, then, only an initial frame sync signal is required to initiate the multipacket transfer. The ’C5x does support operation of the serial port in this fashion, referred to as continuous mode, which is selected by clearing the FSM bit in the SPC to 0. Continuous mode serial port operation is described in detail in subsection 9.7.5, *Continuous Mode Transmit and Receive Operations*.

9.7.5 Continuous Mode Transmit and Receive Operations

In continuous mode, a frame sync on FSX/FSR is not necessary for consecutive packet transfers at maximum packet frequency after the initial pulse. Continuous mode is selected by setting FSM = 0. Note that when FSM = 0, frame sync pulses are not required, but they are not ignored, therefore, improperly timed frame syncs may cause errors in serial transfers. Serial port operation under various error conditions is described in detail in subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46.

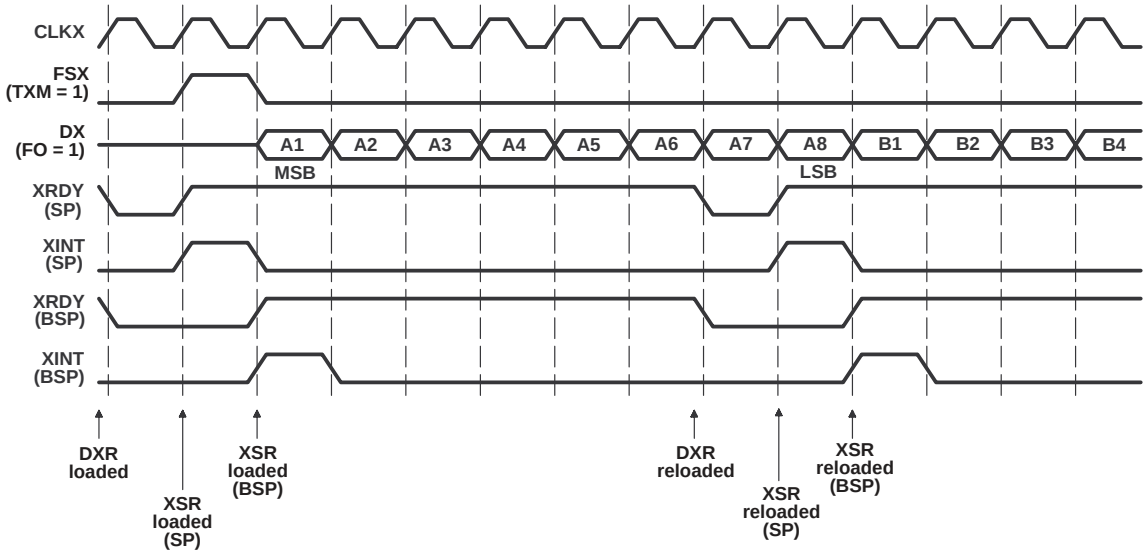
In continuous mode transmission, one frame sync is generated following the first DXR load, and no further frame syncs are generated. As long as DXR is reloaded once every transmission, continuous transfers are maintained. Failing to update DXR causes the serial port to halt, as in the burst mode case ($\overline{\text{XSREMPTY}}$ becomes asserted, etc). If DXR is reloaded after a halt, the device begins continuous mode transmission again and generates a single FSX, assuming that internal frame sync generation is selected.

The distinction between internal and external frame syncs for continuous mode is similar to that of burst mode, as discussed in subsection 9.7.4, *Burst Mode Transmit and Receive Operations*. If frame sync is externally generated (TXM = 0), then when DXR is loaded, the appearance of the frame sync pulse initiates continuous mode transmission. Continuous mode transmission may be discontinued (and burst mode resumed) only by reconfiguring and resetting the serial port (see subsection 9.7.2, *Serial Port Interface Operation*, on page 9-25). Simply changing the FSM bit during transmit or halt will not properly switch to burst mode.

Continuous mode transmit timing, shown in Figure 9–25, is similar to maximum packet frequency transmission in burst mode as shown in Figure 9–23. The major difference is the lack of a frame sync pulse after the initial one. As long as DXR is updated once per transmission, this mode will continue. Overwrites to DXR behave just as in burst mode; the last data written will be

transmitted. XSR operation is the same as in burst mode. A new external FSX pulse will abort the present transmission, cause one data packet to be lost, and initiate a new continuous mode transmit. This is explained in more detail in subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46.

Figure 9–25. Continuous Mode Serial Port Transmit



Continuous mode reception is similar to the transmit operation. After the initial frame sync pulse on FSR, no further frame syncs are required. This mode will continue as long as DRR is read every transmission. If DRR is not read by the end of the next transfer, the receiver will halt, and RSRFULL is set, indicating overrun. See subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46.

Overrun in continuous mode effects the SP and the BSP differently. On the SP, once overrun has occurred, reading DRR will restart continuous mode at the next word/byte boundary after DRR is read; no new FSR pulse is required. On the BSP, continuous mode reception does not resume until DRR is read and an FSR occurs.

Continuous mode reception may only be discontinued by reconfiguring and re-setting the serial port. Simply changing the FSM bit during a reception or halt will not properly switch to burst mode. Continuous mode receive timing is shown in Figure 9–26.

Figure 9–26. Continuous Mode Serial Port Receive

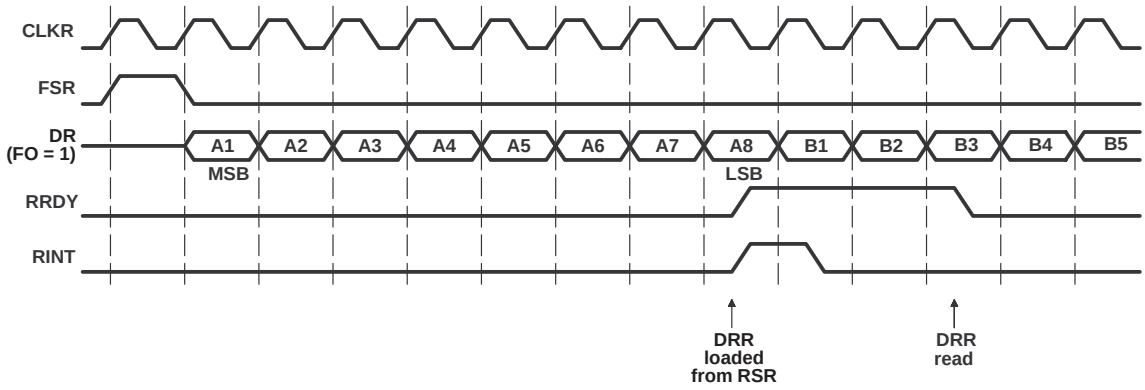


Figure 9–26 shows only one frame sync pulse; otherwise, it is similar to Figure 9–24. If a pulse occurs on FSR during a transfer (an error), then the receive operation is aborted, one packet is lost, and a new receive cycle is begun. This is discussed in more detail in subsection 9.7.2, *Serial Port Interface Operation*, on page 9-25 and in subsection 9.7.6, *Serial Port Interface Exception Conditions*.

9.7.6 Serial Port Interface Exception Conditions

Exception (or error) conditions result from an unexpected event occurring on the serial port. These conditions are operational aberrations such as overrun, underflow, or a frame sync pulse during a transfer. Understanding how the serial port handles these errors and the state it acquires during these error conditions is important for efficient use of the serial port. Because the error conditions differ slightly in burst and continuous modes, they are discussed separately.

Burst Mode

In burst mode, one type of error condition (presented in subsection 9.7.2, *Serial Port Interface Operation*) is receive overrun, indicated by the RSRFULL flag. This flag is set when the device has not read incoming data and more data is being sent. If this condition occurs, the processor halts serial port receives until DRR is read. Thus, any further data sent may be lost.

Overrun is handled differently on the SP and on the BSP. On the SP, the contents of RSR are preserved on overrun, but since RSRFULL is not set to 1 until the next FSR occurs after the overflowing reception, incoming data usually begins being lost as soon as RSRFULL is set. Data loss can only be avoided if RSRFULL is polled in software and the DRR is read immediately after

RSRFULL is set to 1. This is normally possible only if the CLKR frequency is slow with respect to CLKOUT1, since RSRFULL is set on the falling edge of CLKR during FSR, and data begins being received on the following rising edge of CLKR. The time available for polling RSRFULL and reading the DRR to avoid data loss is, therefore, only half of one CLKR cycle.

On the BSP, RSRFULL is set on the last valid bit received, but the contents of RSR are never transferred to DRR, therefore, the complete transferred word in RSR is lost. If the DRR is read (clearing RSRFULL) before the next FSR occurs, subsequent transfers can be received properly.

Another type of receive error is caused if frame sync occurs during a receive (that is, data is being shifted into RSR from DR). If this happens, the present receive is aborted and a new one begins. Thus, the data that was being loaded into RSR is lost, but the data in DRR is not (no RSR-to-DRR copy occurs). Burst mode serial port receiver behavior under normal and error conditions for the SP is shown in Figure 9–27 and for the BSP is shown in Figure 9–28.

Figure 9–27. SP Receiver Functional Operation (Burst Mode)

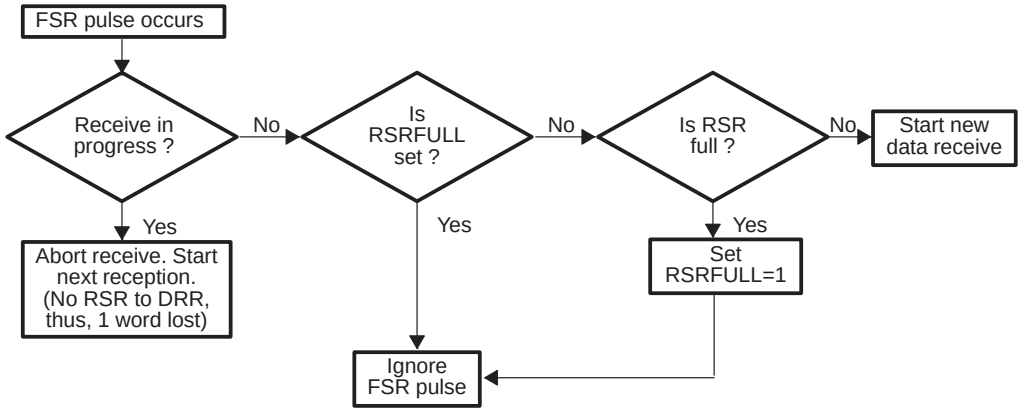
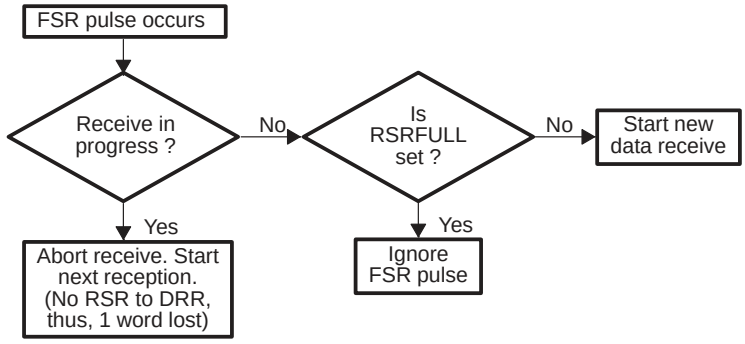
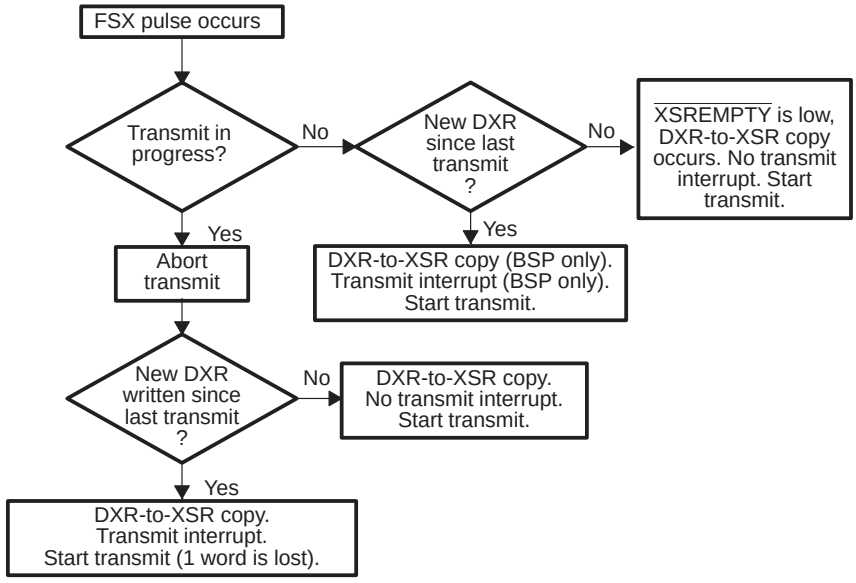


Figure 9–28. BSP Receiver Functional Operation (Burst Mode)



Transmitter exception conditions in burst mode may occur for several possible reasons. Underflow, which is described in subsection 9.7.3, *Setting the Serial Port Configuration*, on page 9-27 is an exception condition that may occur in burst mode, however, underflow is not normally considered an error. An exception condition that causes errors in transmitted data occurs when frame sync pulses occur at inappropriate times during a transfer. If a transmit is in progress (that is, XSR data is being driven on DX) when a frame sync pulse occurs, the transmission is aborted, and the data in XSR is lost. Then, whatever data is in DXR at the time of the frame sync pulse is transferred to XSR (DXR-to-XSR copy) and is transmitted. Note, however, that in this case an XINT is generated only if the DXR has been written to since the last transmit. Also, if $\overline{\text{XSREMPTY}}$ is active and a frame sync pulse occurs, the old data in DXR is shifted out. Figure 9–29 summarizes serial port transmit behavior under error and nonerror conditions. Note that if an FSX occurs when no transmit is in progress, and DXR has been reloaded since the last transmit, the DXR-to-XSR copy and generation of transmit interrupt occur at this point only on the BSP. On the SP, these two events occur at the time the DXR was reloaded.

Figure 9–29. SP/BSP Transmitter Functional Operation (Burst Mode)

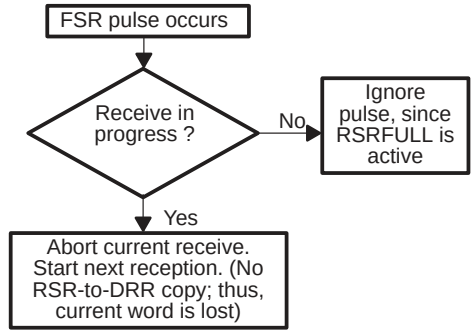


Continuous Mode

In continuous mode, errors take on a broader meaning, since data transfer is intended to occur at all times. Thus, underflow ($\overline{\text{XSREMPTY}}$ = 0) constitutes an error in continuous mode because data will not be transmitted. As in burst mode, overrun (RSRFULL = 1) is also an error, and in continuous mode, both overrun and underflow cause the serial port receive or transmit sections, respectively, to halt (see subsection 9.7.3, *Setting the Serial Port Configuration*, on page 9-27 for a description of these conditions). Fortunately, underflow and overrun errors may not be catastrophic; they can often be corrected simply by reading DRR or writing to DXR.

The SP and the BSP are affected differently when overrun occurs in continuous mode. In the SP, when DRR is read to deactivate RSRFULL, a frame sync pulse is not required in order to resume continuous mode operation. The receiver keeps track of the transfer word boundary, even though it is not receiving data. Therefore, when the RSRFULL flag is deactivated by a read from DRR, the receiver begins reading from the correct bit. On the BSP, since an FSR pulse is required to restart continuous reception, this also reestablishes the proper bit alignment, in addition to restarting reception. Figure 9–30 shows receiver functional operation in continuous mode.

Figure 9–30. SP/BSP Receiver Functional Operation (Continuous Mode)

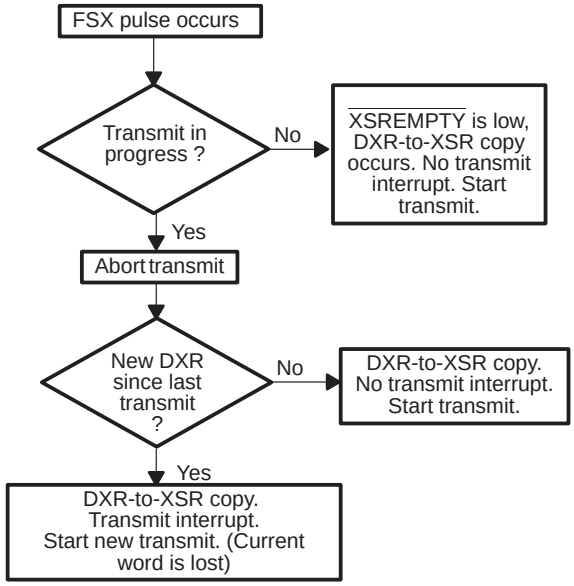


During a receive in continuous mode, if a frame sync pulse occurs, this causes a receive abort condition, and one packet of data is lost (this is caused because the frame sync pulse resets the RSR bit counter). The data present on DR then begins being shifted into RSR, starting again from the first bit. Note that if a frame sync occurs after deactivating the RSRFULL flag by reading DRR, but before the beginning of the next word boundary, this also creates a receive abort condition.

Another cause for error is the appearance of extraneous frame syncs during a transmission. After the initial frame sync in continuous mode, no others are required; if an improperly timed frame sync pulse occurs during a transmit, the current transfer (that is, serially driving XSR data onto DX) is aborted, and data in XSR is lost. A new transmit cycle is initiated, and transfers continue as long as the DXR is updated once per transmission afterward. Figure 9–31 shows continuous mode transmitter functional operation.

Note that if $\overline{\text{XSREMPTY}}$ is active in continuous mode and an external frame sync occurs, the previous DXR data is transmitted as in burst mode operation.

Figure 9–31. SP/BSP Transmitter Functional Operation (Continuous Mode)



9.7.7 Example of Serial Port Interface Operation

The following two code examples illustrate a one-way transmit from device 0 to device 1 of an arithmetic sequence of numbers. The numbers are written in each device in a block from 9000h to B000h in data memory. Device 0 waits in a BIO loop for a ready-to-receive signal (XF) from device 1 and initializes the transfer with a value of 0. Both routines assume that the DP is cleared to 0, and that ARP = 7.

Example 9–3 shows the code running on device 0. Only its transmit interrupt (XINT) is enabled; its transmit ISR writes the value it will send into its own memory.

Example 9–3. Device 0 Transmit Code (Serial Port Interface Operation)

```
* Device 0 – Transmit side
:         :                    :
                               ;Setup SPC as CLK source
                               ;and internal frame sync
          SPLK   #0038h, SPC   ;Set TXM=MCM=FSM=1,
                               ;TDM=DLB=FO=0.
                               ;And put SP into reset
                               ;(XRST=RRST=0)
          SPLK   #00F8h, SPC   ;Take SP out of reset
                               ;Setup interrupts
          SPLK   #0ffffh, IFR  ;clear IFR
          SPLK   #020h, IMR    ;Turn on XINT
          CLRC   INTM          ;enable interrupts
ILOOP     BCND   SENDZ, BIO    ;Wait for ready–to–receive
          B      ILOOP         ;from other device
SENDZ     LACL   #0            ;First transmit/write
                               ;value is 0
          LAR    AR7, #9000h   ;Setup where to write
          SACL   *             ;Write first value
          SACL   DXR           ;Transmit first value
SELF1     B      SELF1         ;Wait for interrupts
XMT_ISR   LACC   AR7           ;Check if past 0x0b000
          SUB    #0B000h       ;i.e. end of block
          BCND   END_SERP,GEQ  ;Go to tight loop if so
                               ;Add one and transmit
          LACL   *+            ;Load value
          ADD    #1            ;Add one
          SACL   *             ;Write value
          SACL   DXR           ;Transmit value
          RETE
END_SERP  B      END_SERP      ;Sit in tight loop after
                               ;block is complete.
```

Example 9–4 shows the code for device 1. It sends a ready-to-receive signal (XF) to device 0. Only its receive interrupt (RINT) is enabled, and its receive ISR reads from the DRR, writes to the block, and checks to see if the end of the block has been reached.

Example 9–4. Device 1 Receive Code (Serial Port Interface Operation)

```
Device 1 – Receive
                              ;Set SP as CLK, frame
                              ;sync receive
          SPLK   #0008h, SPC  ;Set TXM=MCM=DLB=FO=0,
                              ;FSM=1.
                              ;And put SP into reset
                              ;(XRST=RRST=0)
          SPLK   #00C8h, SPC  ;Take SP out of reset
                              ;Setup interrupts
          SPLK   #0ffffh, IFR ;clear IFR
          SPLK   #010h, IMR   ;Turn on RINT
          CLRC   INTM         ;Enable interrupts
          LAR    AR7, #9000h  ;Setup where to write
                              ;received data
          CLRC XF
                              ;Signal ready to receive
SELF1     B      SELF1        ;Wait for interrupts
RCV_ISR
          LACL   DRR          ;Load received value
          SACL   *+           ;Write to memory block
          LACC   AR7          ;Check if past 0x0b000
          SUB    #0b000h      ;i.e. end of block
          BCND   END_SERP,GEQ ;Go to tight loop if so
END_SERP  B      END_SERP     ;Sit in tight loop after
                              ;block is complete.
```

9.8 Buffered Serial Port (BSP) Interface

The buffered serial port (BSP) is made up of a full-duplex, double-buffered serial port interface, which functions in a similar manner to the ’C5x standard serial port (SP), and an autobuffering unit (ABU) (see Figure 9–32). The SP section of the BSP is an enhanced version of the ’C5x standard serial port as implemented on the ’C50, ’C51, ’C52, and ’C53. The ABU is an additional section of logic which allows the SP section to read/write directly to ’C5x internal memory independent of the CPU. This results in a minimum overhead for serial port transfers and faster data rates. The BSP is available on the ’LC56, ’LC57, and ’C57S devices.

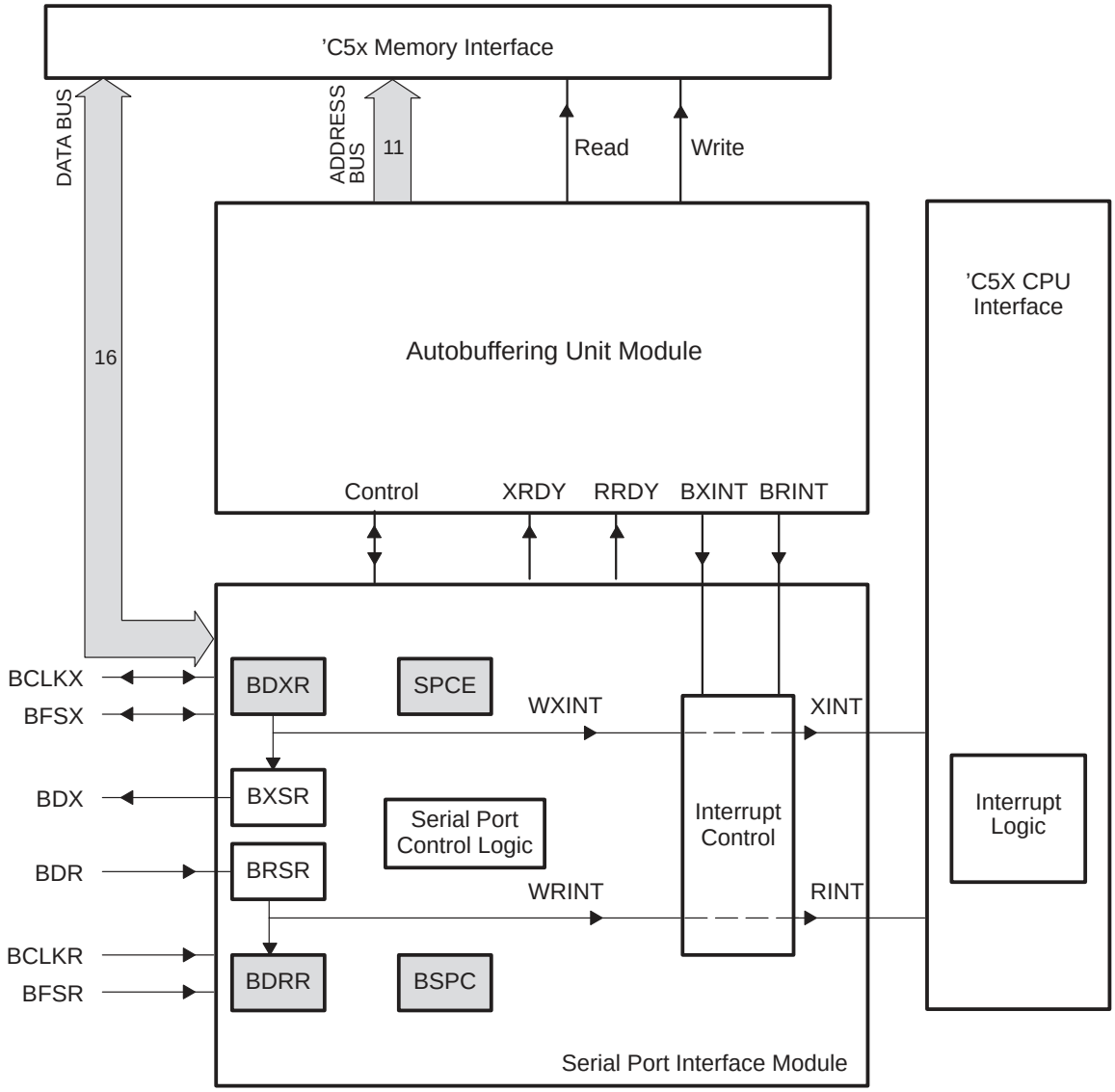
The full duplex BSP serial interface provides direct communication with serial devices such as codecs, serial A/D converters, and other serial devices with a minimum of external hardware. The double-buffered BSP allows transfer of a continuous communication stream in 8-,10-,12- or 16-bit data packets. Frame synchronization pulses as well as a programmable frequency serial clock can be provided by the BSP for transmission and reception. The polarity of frame sync and clock signals are also programmable. The maximum operating frequency is CLKOUT1 (28.6M bps at 35ns, 40M bps at 25 ns). The BSP transmit section includes a pulse coded modulation (PCM) mode that allows easy interface with a PCM line. Operation of the BSP in standard (nonbuffered) mode is detailed in subsection 9.8.1 on page 9-55.

The ABU has its own set of circular addressing registers, each with corresponding address generation units. Memory for transmit and receive buffers resides within a special 2K word block of ’C5x internal memory. This memory can also be used by the CPU as general purpose storage, however, this is the only memory block in which autobuffering can occur.

Using autobuffering, word transfers occur directly between the SP section and the ’C5x internal memory automatically using the ABU embedded address generators. The length and starting addresses of the buffers within the 2K block are programmable, and a buffer empty/full interrupt can be generated to the CPU. Buffering can easily be halted using the autodisabling capability. ABU operation is detailed in subsection 9.8.2 on page 9-60.

The BSP autobuffering capability can be separately enabled for the transmit and receive sections. When autobuffering is disabled (standard mode), data transfers with the SP section occur under software control in the same fashion as with the standard ’C5x serial port. In this mode, the ABU is transparent, and the WXINT and WRINT interrupts generated each time a word is transmitted or received are sent to the CPU as transmit interrupt (XINT) and receive interrupt (RINT). When autobuffering is enabled, the XINT and RINT interrupts are only generated to the CPU each time half of the buffer is transferred.

Figure 9–32. BSP Block Diagram



Most aspects of BSP operation are similar to that of the 'C5x standard serial port. Section 9.7, *Serial Port Interface*, on page 9-23 discusses operation of both the 'C5x standard serial port and the BSP in standard mode. Since standard mode BSP operation is a superset of standard SP operation, Section 9.7 should first be studied before the rest of this section is read.

System considerations of BSP operation such as initialization and low power modes are discussed in subsection 9.8.3 on page 9-69.

9.8.1 BSP Operation in Standard Mode

BSP operation in standard mode is discussed in Section 9.7, *Serial Port Interface*, on page 9-23. This subsection summarizes the differences between SP operation and standard mode BSP operation. The enhanced BSP features are available both in standard mode and in autobuffering mode. ABU is discussed in subsection 9.8.2 on page 9-60. Information presented in this section assumes familiarity with standard mode operation as described in Section 9.7.

The BSP uses its own dedicated memory-mapped data transmit, data receive and serial port control registers (BDXR, BDRR, and BSPC). The BSP also utilizes an additional control register, the BSP control extension register (SPCE), in implementing its enhanced features and controlling the ABU. The BDRR, BDXR, and BSPC registers function similarly to their counterparts in the SP as described in Section 9.7. As with the SP, the BSP transmit and receive shift registers (BXSR and BRSR) are not directly accessible in software but facilitate the double-buffering capability. If the serial port is not being used, the BDXR and the BDRR registers can be used as general purpose registers. In this case, BFSR should be set to an inactive state to prevent a possible receive operation from being initiated. Note, however, that program access to BDXR or BDRR is limited when autobuffering is enabled for transmit or receive, respectively. BDRR can only be read, and BDXR can only be written when the ABU is disabled. BDRR can only be written when the BSP is in reset. BDXR can be read any time.

The buffered serial port registers are summarized in Table 9–15. The ABU utilizes several additional registers which are discussed in subsection 9.8.2, *Autobuffering Unit (ABU) Operation*, on page 9-60.

Table 9–15. Buffered Serial Port Registers

Address	Register	Description
0030h	BDRR	16-bit BSP data receive register
0031h	BDXR	16-bit BSP data transmit register
0032h	BSPC	16-bit BSP control register
0033h	SPCE	16-bit BSP control extension register
—	BRSR	16-bit BSP data receive shift register
—	BXSR	16-bit BSP data transmit shift register

9.8.1.1 Differences Between SP and BSP Operation in Standard Mode

The differences between SP and BSP operation in standard mode are discussed in detail in the standard mode serial port operation (Section 9.7 on page 9-23). These differences relate primarily to boundary conditions, however, in some systems, these differences may be significant. The differences are summarized in Table 9–16.

Table 9–16. Differences Between SP and BSP Operation in Standard Mode

Condition	SP	BSP
RSRFULL is set.	RSRFULL is set when RSR is full and then an FSR occurs, except in continuous mode where RSRFULL is set as soon as RSR is full.	RSRFULL is set as soon as BRSR is full.
Preservation of data in RSR on overrun.	RSR contents are preserved on overrun.	BRSR contents are not preserved on overrun.
Continuous mode receive restart after overrun.	Receive restarts as soon as DRR is read (see subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46).	Receive does not restart until BDRR is read and then a BFSR occurs.
Sign extension in DRR on 8-, 10-, or 12-bit transfers.	No	Yes
XSR load, $\overline{\text{XSREMPTY}}$ clear, XRDY/XINT generation.	Occur when DXR is loaded.	Occur when when a BFSX occurs after BDXR is loaded.
Program accessibility to DXR and DRR.	DRR and DXR can be read or written under program control at any time. Note that caution should be exercised when reads and writes of the DRR may be close in time to serial port receptions. In this case, a DRR read may not yield the result that was previously written by the program. Also note that rewrites of DXR may cause loss (and therefore non-transmission) of previously written data depending on the relative timing of the writes and FSX (see subsection 9.7.4, *Burst Mode Transmit and Receive Operations*, on page 9-37).	BDRR can only be read and BDXR can only be written when the ABU is disabled. BDRR can only be written when the BSP is in reset. BDXR can be read any time. The same precautions with regard to reads and writes to these registers apply as in SP.
Maximum serial port clock rate.	CLKOUT1/2	CLKOUT1

Table 9–16. Differences Between SP and BSP Operation in Standard Mode (Continued)

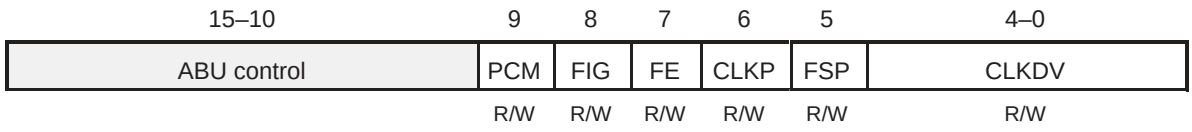
Condition	SP	BSP
Initialization timing requirements.	On the SP, the serial port may be taken out of reset at any time with respect to FSX/FSR, however, if $\overline{\text{XRST}}$/$\overline{\text{RRST}}$ go high during or after the frame sync, the frame sync may be ignored.	On the BSP, exiting serial port reset under certain conditions must precede FSX timing by one CLKOUT1 cycle in standard mode and by six CLKOUT1 cycles in autobuffering mode (see subsection 9.8.3, *System Considerations of BSP Operation*, on page 9-69).
Operates in IDLE2 mode.	No	Yes (see subsection 9.8.3, *System Considerations of BSP Operation*, on page 9-69).

9.8.1.2 Enhanced BSP Features

The enhanced features that the BSP offers include the capability to generate programmable rate serial port clocks, select positive or negative polarities for clock and frame sync signals, and to perform transfers of 10- and 12-bit words, in addition to the 8- and 16-bit transfers offered by the SP. Additionally, the BSP implements the capability to specify that frame sync signals be ignored until instructed otherwise, and provides a dedicated operating mode which facilitates its use with PCM interfaces.

The SPCE contains the control and status bits that are used in the implementation of these enhanced BSP features and the ABU. The 10 LSBs of SPCE are dedicated to the enhanced features control, whereas the 6 MSBs are used for ABU control, which is discussed in subsection 9.8.2, *Autobuffering Unit (ABU) Operation*, on page 9-60. Figure 9–33 shows the SPCE bit positions and Table 9–17 summarizes the function of the SPCE bits. The value of the SPCE upon reset is 3. This results in standard mode operation compatible with the SP.

Figure 9–33. BSP Control Extension Register (SPCE) Diagram — Serial Port Control Bits



Note: R = Read, W = Write

Table 9–17. BSP Control Extension Register (SPCE) Bit Summary — Serial Port Control Bits

Bit	Name	Reset value	Function
15–10	ABU control	—	Reserved for autobuffering unit control (see subsection 9.8.2, *Autobuffering Unit (ABU) Operation*, on page 9-60).
9	PCM	0	Pulse Code Modulation Mode. This control bit puts the serial port in pulse-code modulation (PCM) mode. The PCM mode only affects the transmitter. BDXR-to-BXSR transfer is not affected by the PCM bit value. PCM = 0 Pulse-coded modulation mode is disabled. PCM = 1 Pulse-coded modulation mode is enabled. In PCM mode, BDXR is transmitted only if its most significant B bit is set to 0. If this bit is set to 1, BDXR is not transmitted and BDX is put in high impedance during the transmission period.
8	FIG	0	Frame Ignore. This control bit operates only in transmit continuous mode with external frame and in receive continuous mode. FIG = 0 Frame sync pulses following the first frame pulse restart the transfer. FIG = 1 Frame sync pulses following the first frame pulse that initiates a transfer operation are ignored.
7	FE	0	Format Extension. The FE bit in conjunction with FO in the SPC register (Table 9–13 on page 9-28) specifies the word length. When FO FE = 00, the format is 16-bit words; when FO FE = 01, the format is 10-bit words; when FO FE = 10, the format is 8-bit words; and when FO FE = 11, the format is 12-bit words. Note that for 8-, 10-, and 12-bit words, the received words are right justified and the sign bit is extended to form a 16-bit word. Words to transmit must be right justified. See Table 9–18 for the word length configurations.
6	CLKP	0	Clock Polarity. This control bit specifies when the data is sampled by the receiver and transmitter. CLKP = 0 Data is sampled by the receiver on BCLKR falling edge and sent by the transmitter on BCLKX rising edge. CLKP = 1 Data is sampled by the receiver on BCLKR rising edge and sent by the transmitter on BCLKX falling edge.

Table 9–17. BSP Control Extension Register (SPCE) Bit Summary — Serial Port Control Bits (Continued)

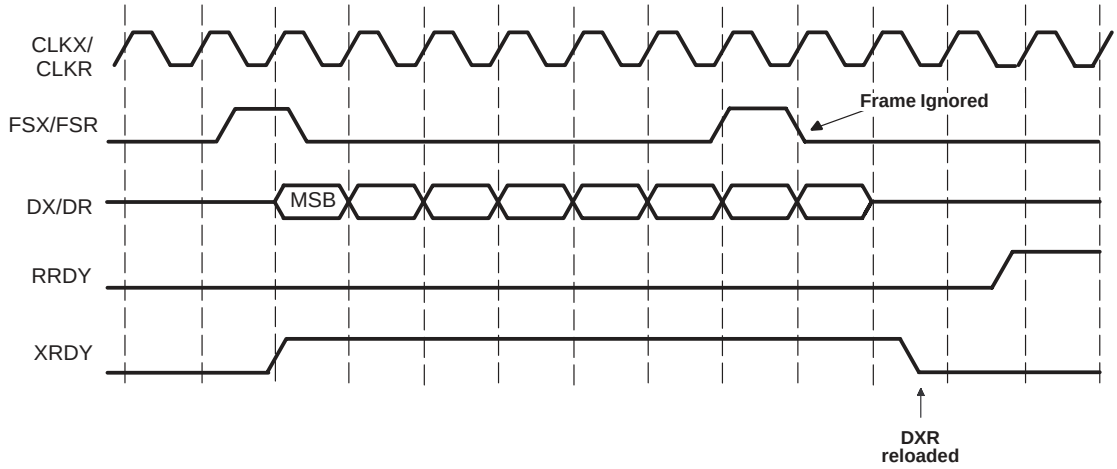
Bit	Name	Reset value	Function
5	FSP	0	Frame Sync Polarity. This control bit specifies whether frame sync pulses (BFSX and BFSR) are active high or low. FSP = 0 Frame sync pulses (BFSX and BFSR) are active high. FSP = 1 Frame sync pulses (BFSX and BFSR) are active low.
4–0	CLKDV	00011	Internal Transmit Clock Division factor. When the MCM bit of BSPC is set to 1, CLKX is driven by an on-chip source having a frequency equal to 1/(CLKDV+1) of CLKOUT. CLKDV range is 0–31. When CLKDV is odd or equal to 0, the CLKX duty cycle is 50%. When CLKDV is an even value (CLKDV=2p), the CLKX high and low state durations depend on CLKP. When CLKP is 0, the high state duration is p+1 cycles and the low state duration is p cycles; when CLKP is 1, the high state duration is p cycles and the low state duration is p+1 cycles.

Table 9–18. Buffered Serial Port Word Length Configuration

FO	FE	Buffered Serial Port Word Length Configuration
0	0	16-bit words transmitted and received. (Reset values)
0	1	10-bit words transmitted and received.
1	0	8-bit words transmitted and received.
1	1	12-bit words transmitted and received.

These enhanced features allow greater flexibility in serial port interface in a variety of areas. In particular, the frame ignore feature offers a capability which allows a mechanism for effectively compressing transferred data packets if they are not transferred in 16 bit format. This feature is used with continuous receptions and continuous transmits with external frame sync. When FIG=0, if a frame sync pulse occurs after the initial one, the transfer is restarted; when FIG=1, this frame sync is ignored. Setting FIG to 1 allows, for example, effectively achieving continuous 16-bit transfers under circumstances where frame sync pulses occur every 8-, 10- or 12-bits. Without using FIG, each transfer of less than 16 bits requires an entire 16-bit memory word, and each 16 bits transferred as two 8-bit bytes requires two memory words and two transfer operations, rather than one of each. Using FIG, therefore, can result in a significant improvement in buffer size requirement in both autobuffered and standard mode, and a significant improvement in CPU cycle overhead required to handle serial port transfers in standard mode. Figure 9–34 shows an example with the BSP configured in 16-bit format but with a frame sync after 8 bits.

Figure 9–34. Transmit Continuous Mode with External Frame and FIG = 1 (Format is 16 bits)



9.8.2 Autobuffering Unit (ABU) Operation

Since ABU functionality is a superset of standard mode serial port operation, Section 9.7, *Serial Port Interface*, on page 9-23 and subsection 9.8.1, *BSP Operation in Standard Mode,* on page 9-55 should first be studied before this subsection is read. Also, note that when operating in autobuffering mode, the serial port control and status bits in BSPC and SPCE function in the same fashion as in standard mode.

The ABU implements the capability to move data transferred on the serial port to and from internal 'C5x memory independent of CPU intervention.

The ABU utilizes five memory-mapped registers: the address transmit register (AXR), the block size transmit register (BKX), the address receive register (ARR), and the block size receive register (BKR), along with the SPCE. These registers are summarized in Table 9–19.

Table 9–19. Autobuffering Unit Registers

Address	Register	Description
0033h	SPCE	16-bit BSP control extension register
0034h	AXR	11-bit BSP address transmit register (ABU)
0035h	BKX	11-bit BSP transmit buffer size register (ABU)
0036h	ARR	11-bit BSP address receive register (ABU)
0037h	BKR	11-bit BSP receive buffer size register (ABU)

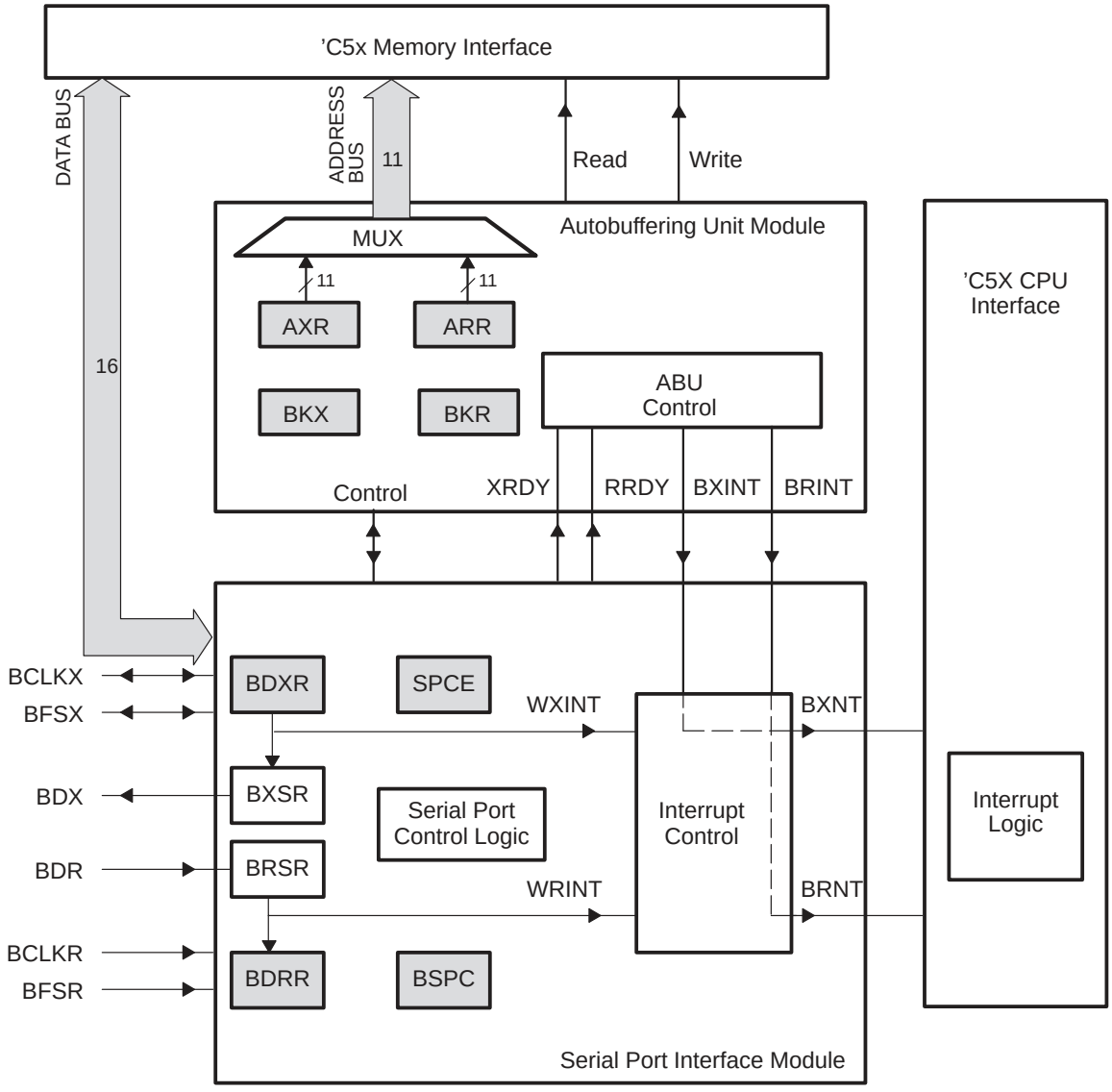
Figure 9–35 shows the block diagram of the ABU. The SPCE contains bits which control ABU operation and will be discussed in detail later in this subsection. AXR, BKX, ARR, and BKR, along with their associated circular addressing logic, allow address generation for accessing words to be transferred between the ’C5X internal memory and the BSP data transmit register (BDXR) and BSP data receive register (BDRR) in autobuffering mode. The address and block size registers as well as circular addressing are also discussed in detail later in this subsection.

Note that the 11-bit memory mapped AXR, BKX, ARR, and BKR registers are read as 16-bit words, with the five most significant bits read as zeroes and the 11-bit register contents right justified in the least significant 11 bits. If autobuffering is not used, these registers can be used for general purpose storage of 11-bit data.

The transmit and receive sections of the ABU can be enabled separately. When either section is enabled, access to its corresponding serial port data register (BDXR or BDRR) through software is limited. The BDRR can only be read, and the BDXR can only be written when the ABU is disabled. The BDRR can only be written when the BSP is in reset. The BDXR can be read any time. When either transmit or receive autobuffering is disabled, that section operates in standard mode, and its portion of the ABU is transparent.

The ABU also implements the capability to generate CPU interrupts when transmit and receive buffers have been halfway or entirely filled or emptied. These interrupts take the place of the transmit and receive interrupts in standard mode operation, which are not generated in autobuffering mode. This mechanism features an autodisabling capability which can be used to automatically terminate autobuffering when either the half-of- or bottom-of-buffer boundary is crossed. These features are also described in detail later in this subsection.

Figure 9–35. ABU Block Diagram



Burst or continuous mode, as described in Section 9.7, *Serial Port Interface*, can be used in conjunction with the autobuffering capability. Note that due to the nature of autobuffering mode, however, if burst mode with internal frame sync is selected, this will effectively result in continuous transmission with FSX generated by the BSP at the start of each transmission.

The internal ’C5X memory used for autobuffering consists of a 2K-word block of single-access memory that can be configured as data, program, or both (as with other single-access memory blocks). This memory can also be used by the CPU as general purpose storage, however, this is the only memory block in which autobuffering can occur. Since the BSP is implemented on several different TMS320 devices, the actual base address of the ABU memory may not be the same in all cases. The 2K-word block of BSP memory is lcoated at 800h–FFFh in data memory or at 8000h–87FFh in program memory as specified by the RAM and OVLY control bits.

When the ABU is enabled, this 2K-word block of memory can still be accessed by the CPU within data and/or program spaces. Conflicts may therefore occur between the CPU and the ABU if the 2K-word block is accessed at the same time by both. If a conflict does occur, priority is given to the ABU, resulting in the CPU access being delayed by one cycle. Accordingly, the worst case situation is that a CPU access could be delayed one cycle each time the ABU accesses the memory block, that is, for every new word transmitted or received. Note that external DMA can only be performed in the 2K-word block of ABU memory when autobuffering is disabled. Also note that when on-chip program memory is secured using the ROM protection feature, the 2K-word block of ABU memory cannot be mapped to program memory. For further information regarding the ROM protection feature, refer to subsection 8.2.4, *Program Memory Protection Feature*, on page 8-14.

When the ABU is enabled for both transmit and receive, if transmit and receive requests from the serial port interface occur at same time, the transmit request takes priority over the receive request. In this case, the transmit memory access occurs first, delaying the receive memory access by generating a wait state. When the transmit memory access is completed, the receive memory access takes place.

9.8.2.1 Autobuffering Control Register

The most-significant six bits in the SPCE constitute the ABU control register (ABUC). Some of these bits are read only, while others are read/write. Figure 9–36 shows the ABUC bit positions and Table 9–20 summarizes the function of each ABUC bit in the SPCE. The value of the SPCE upon reset is 3.

Figure 9–36. BSP Control Extension Register (SPCE) Diagram — ABU Control Bits

15	14	13	12	11	10	9–0
HALTR	RH	BRE	HALTX	XH	BXE	SP control
R/W	R	R/W	R/W	R	R/W	

Note: R = Read, W = Write

Table 9–20. BSP Control Extension Register (SPCE) Bit Summary — ABU Control Bits

Bit	Name	Reset value	Function
15	HALTR	0	Autobuffering Receive Halt. This control bit determines whether autobuffering receive is halted when the current half of the buffer has been received.
			HALTR = 0 — Autobuffering continues to operate when the current half of the buffer has been received.
			HALTR = 1 — Autobuffering is halted when the current half of the buffer has been received. When this occurs, the BRE bit is cleared to 0 and the serial port continues to operate in standard mode.
14	RH	0	Receive Buffer Half Received. This read-only bit indicates which half of the receive buffer has been filled. Reading RH when the RINT interrupt occurs (seen either as a program interrupt or by polling IFR) is a convenient way to identify which boundary has just been crossed.
			RH = 0 — The first half of the buffer has been filled and that receptions are currently placing data in the second half of the buffer.
			RH = 1 — The second half of the buffer has been filled and that receptions are currently placing data in the first half of the buffer.
13	BRE	0	Autobuffering Receive Enable. This control bit enables autobuffering receive.
			BRE = 0 — Autobuffering is disabled and the serial port interface operates in standard mode.
			BRE = 1 — Autobuffering is enabled for the receiver.
12	HALTX	0	Autobuffering Transmit Halt. This control bit determines whether autobuffering transmit is halted when the current half of the buffer has been transmitted.
			HALTX = 0 — Autobuffering continues to operate when the current half of the buffer has been transmitted.
			HALTX = 1 — Autobuffering is halted when the current half of the buffer has been transmitted. When this occurs, the BXE bit is cleared to 0 and the serial port continues to operate in standard mode.

Table 9–20. BSP Control Extension Register (SPCE) Bit Summary — ABU Control Bits (Continued)

Bit	Name	Reset value	Function
11	XH	0	Transmit Buffer Half Transmitted. This read-only bit indicates which half of the transmit buffer has been transmitted. Reading XH when the XINT interrupt occurs (seen either as a program interrupt or by polling IFR) is a convenient way to identify which boundary has just been crossed. XH = 0 The first half of the buffer has been transmitted and transmissions are currently taking data from the second half of the buffer. XH = 1 The second half of the buffer has been transmitted and transmissions are currently taking data from the first half of the buffer.
10	BXE	0	Autobuffering Transmit Enable. This control bit enables the autobuffering transmit. BXE = 0 Autobuffering is disabled and the serial port operates in standard mode. BXE = 1 Autobuffering is enabled for the transmitter.
9–0	SP control	—	Serial Port Interface Control bits (see subsection 9.8.1.2, *Enhanced BSP Features*, on page 9-57).

9.8.2.2 Autobuffering Process

The autobuffering process occurs between the ABU and the 2K-word block of ABU memory. Each time a serial port transfer occurs, the data involved is automatically transferred to or from a buffer in the 2K-word block of memory under control of the ABU. During serial port transfers in autobuffering mode, interrupts are not generated with each word transferred as they are in standard mode operation. This prevents the overhead of having the CPU directly involved in each serial port transfer. Interrupts are generated to the CPU only each time one of the half-buffer boundaries is crossed.

Within the 2K-word block of ABU memory, the starting address and size of the buffers allocated is programmable using the 11-bit address registers (AXR and ARR) and the 11-bit block size registers (BKX and BKR). The transmit and receive buffers can reside in independent areas, overlapping areas or the same area, which allows transmitting from a buffer while receiving into the same buffer if desired.

The autobuffering process utilizes a circular addressing mechanism to access buffers within the 2K word block of ABU memory. This mechanism operates in the same fashion for transmit and receive. For each direction (transmit or receive), two registers specify the buffer size and the current address in the buffer. These registers are the block size and address register for transmit and receive. Each of the BK/AR register pairs fully specify the top and bottom of buffer addresses for transmit and receive. Note that this circular addressing mechanism only effects accesses into the 2K word block by the ABU. Accesses to this memory by the CPU are performed strictly according to the addressing mode(s) selected in the assembly language instructions which perform the memory access.

The circular addressing mechanism automatically recirculates ABU memory accesses through the specified buffer, returning to the top of the buffer each time the bottom of the buffer is reached. The circular addressing mechanism is initialized by loading BK with the exact size of the desired buffer (as opposed to size–1) and AR with a value which contains both the base address of the buffer within the 2K word block and the initial starting address within this buffer (this is explained in detail below). Often the initial starting address within the buffer is 0, indicating the start of the buffer (the top-of-buffer address), but the initial starting address may be any point within the defined buffer range.

Once initialized, BK can be considered to consist of two parts; the most significant or higher part (BKH), which corresponds to the all of the most significant 0 bits of BK, and the lower part (BKL), which is the remaining bits, of which the most significant bit is a 1 and whose bit position is designated bit position N. The N bit position also defines the two parts (ARH and ARL) of the address register. The top of buffer address (TBA) is defined by the concatenation of ARH with N+1 least significant 0 bits. The bottom of buffer address (BBA) is defined by the concatenation of ARH and BKL–1, and the current address within the buffer is specified by the complete contents of AR. A circular buffer of size BK must therefore start on an N-bit boundary (the N least significant bits of the address register are 0) where N is smallest integer that satisfies $2^N > BK$, or at the lowest address within the 2K memory block. The buffer consists of two halves, the address range for the first half is:

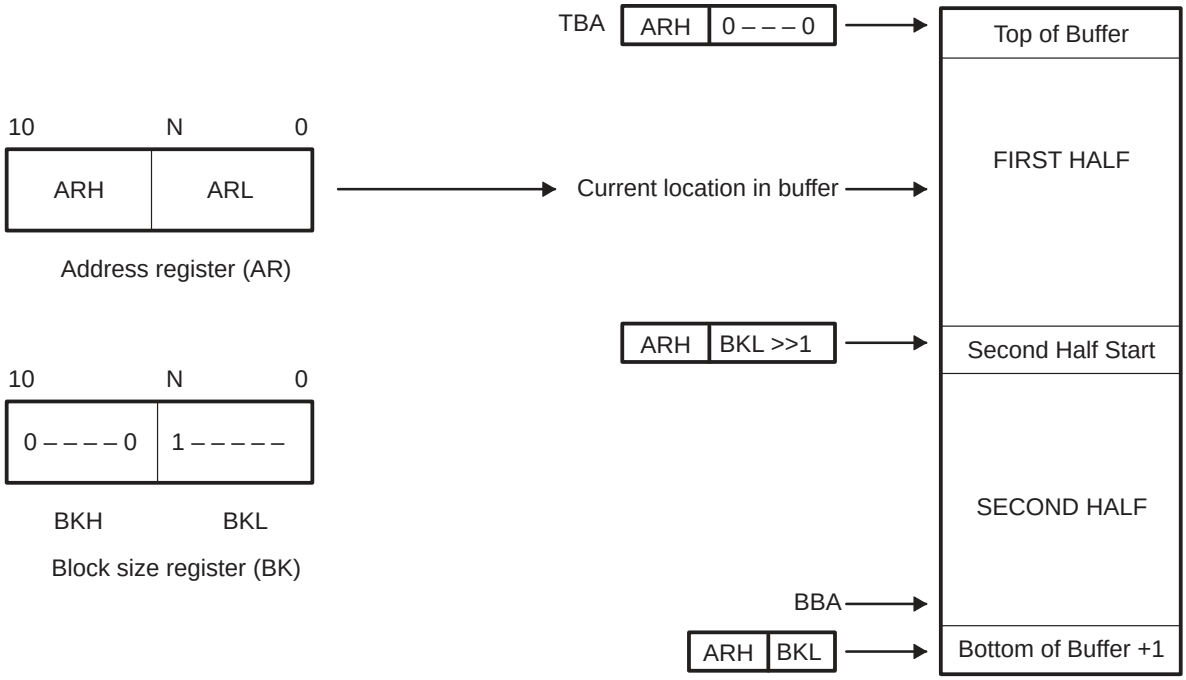
$$ARH|0\cdots0 \text{ to } ARH|[(BKL >> 1) - 1]$$

and the address range for the second half is:

$$ARH|(BKL >> 1) \text{ to } ARH|(BKL - 1)$$

Figure 9–37 illustrates all of the relationships between the defined buffer and the BK and AR registers, the bottom of circular buffer address (BBA), and the top of circular buffer address (TBA).

Figure 9–37. Circular Addressing Registers

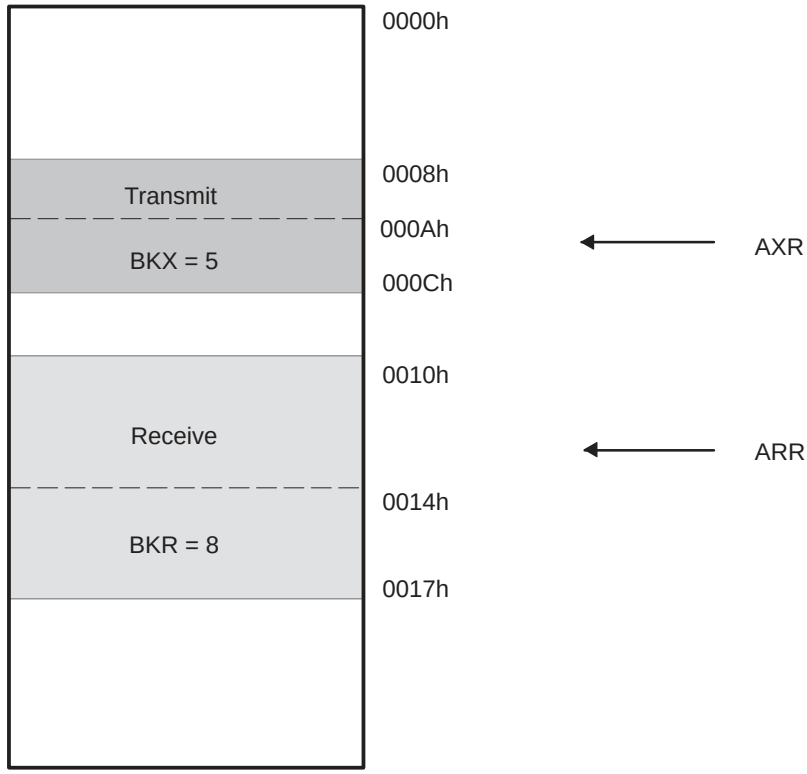


The minimum block size for an ABU buffer is two; the maximum block size is 2047, and any buffer of 2047 to 1024 words must start at a relative address of 0x0000 with respect to the base address of the 2K block of ABU memory. If either of the address registers (AXR or ARR) is loaded with a value specifying a location that is outside the range of the currently allocated buffer size as defined by BK, improper operation may result. Subsequent memory accesses will be performed starting at the location specified, despite the fact that they will be to locations which are outside the range of the desired buffer, and the AR will be incremented with each access until its contents reach the next permitted buffer start address. Any further accesses are then performed using the correct circular buffering algorithm with the new AR contents as the updated buffer start address. It should be noted that any accesses performed with improperly loaded ARs may therefore unexpectedly corrupt some memory locations.

The following example illustrates some of these functional aspects of the auto-buffering process. Consider a transmit buffer of size 5 (BKX = 5) and a receive buffer of size 8 (BKR = 8) as shown in Figure 9–38. The transmit buffer may start at any relative address that is a multiple of 8 (address 0x0000, 0x0008, 0x0010, 0x0018, ···, 0x07F8), and the receive buffer may start at any relative address that is a multiple of 16 (0x0000, 0x0010, 0x0020, ···, 0x07F0). In this

example, the transmit buffer starts at relative address 0x0008 and the receive buffer starts at relative address 0x0010. AXR may therefore contain any value in the range 0x0008–0x000C and ARR may contain any value in the range 0x0010–0x0017. If AXR in this example had been loaded with the value 0x000D (not acceptable in a modulo 5 buffer), memory accesses would be performed and AXR incremented until it reaches address 0x0010 which is an acceptable starting address for a modulo 5 buffer. Note, however, that if this had occurred, AXR would then specify a transmit buffer starting at the same base address as the receive buffer, which may cause improper buffer operation.

Figure 9–38. Transmit Buffer and Receive Buffer Mapping Example



The autobuffering process is activated upon request from serial port interface when XRDY or RRDY goes high, indicating that a word has been received. The required memory access is then performed, following which an interrupt is generated if half of the defined buffer (first or second) has been processed. The RH and XH flags in the SPCE register indicate which half has been processed when the interrupt occurs.

When autodisabling is selected (HALTX or HALTR bit is set), then when the next half (first or second) buffer boundary is encountered, the autobuffering enable bit in the SPCE (BXE or BRE) is cleared so that autobuffering is disabled and does not generate any further requests. When transmit autobuffering is halted, transmission of the current XSR contents and the last value loaded in DXR are completed, since these transfers have already been initiated. Therefore, when using the HALTX function, some delay will normally occur between crossing a buffer boundary and transmission actually stopping. If it is necessary to identify when transmission has actually ended, software should poll for the condition of XRDY = 1 and $\overline{\text{XSREMPTY}}$ = 0, which occurs after last bit has been transmitted.

In the receiver, when using HALTR, since autobuffering is stopped when the most recent buffer boundary is crossed, future receptions may be lost, unless software begins servicing receive interrupts at this point, since BDRR is no longer being read and transferred to memory automatically by the ABU. For explanation of how the serial port operates in standard mode when DRR is not being read, refer to subsection 9.7.6, *Serial Port Interface Exception Conditions*, on page 9-46.

The sequence of events involved in the autobuffering process is summarized as follows:

1) The ABU performs the memory access to the buffer.

2) The appropriate address register is incremented unless the bottom of buffer has been reached, in which case the address register is modified to point to the top of buffer address.

3) Generate an XINT or RINT and update XH/RH if the half buffer or bottom of buffer boundary has been crossed.

4) Autodisable the ABU if this function has been selected and if the half buffer or bottom of buffer boundary has been crossed.

9.8.3 System Considerations of BSP Operation

This subsection discusses several system-level considerations of BSP operation. These considerations include initialization timing issues, software initialization of the ABU, and power down mode operation.

9.8.3.1 Timing of Serial Port Initialization

The ’C5x device utilizes a fully static design, and accordingly, in both the SP and the BSP, serial port clocks need not be running between transfers or prior to initialization. Therefore, proper operation can still result if FSX/FSR occurs

simultaneously with CLKX/CLKR starting. Regardless of whether serial port clocks have been running previously, however, the timing of serial port initialization, and most importantly, when the port is taken out of reset, can be critical for proper serial port operation. The most significant consideration of this is when the port is taken out of reset with respect to when the first frame sync pulse occurs.

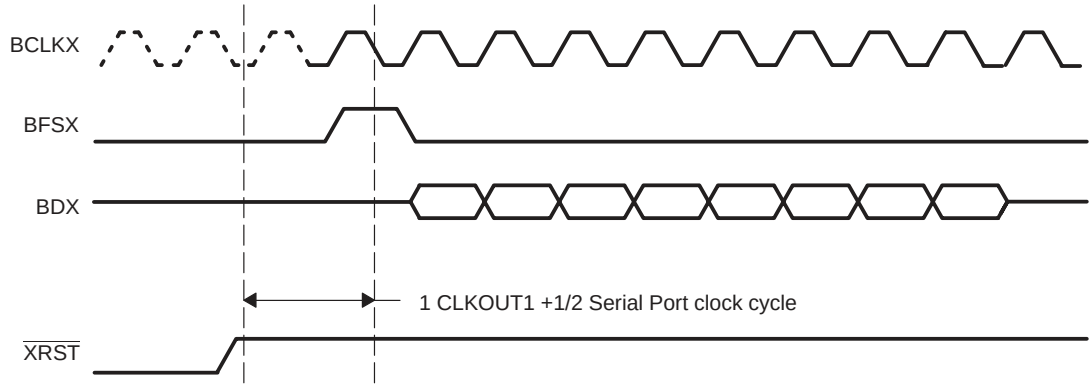
Initialization timing requirements differ on the SP and the BSP. On the SP, the serial port may be taken out of reset at any time with respect to FSX/FSR, however, if $\overline{\text{XRST}}$/$\overline{\text{RRST}}$ go high during or after the frame sync, the frame sync may be ignored. In standard mode operation on the BSP for receive, and for transmit with external frame sync (TXM = 0), the BSP must be taken out of reset at least one full CLKOUT1 cycle plus 1/2 serial port clock cycle prior to the edge of the clock which detects the active frame sync pulse (whether the clock has been running previously or not) for proper operation. See Figure 9–39.

Transmit operations with internal clock and frame sync are not subject to this requirement since frame sync is internally generated automatically (after $\overline{\text{XRST}}$ is cleared (set to 1)) when BDXR is loaded.

Note, however, that if external serial port clock is used with internal frame sync, frame sync generation may be delayed depending on the timing of clearing $\overline{\text{XRST}}$ with respect to the clock.

Figure 9–39 illustrates the standard mode BSP initialization timing requirements for the transmitter. The figure shows standard mode operation with external frame (TXM = 0) and clock (MCM = 0), active high frame sync (FSP = 0) and data samples on rising edge (CLKP = 0). In this example, if the BFSX pulse occurs during the first BCLKX, the transmission is still properly initiated.

Figure 9–39. Standard Mode BSP Initialization Timing



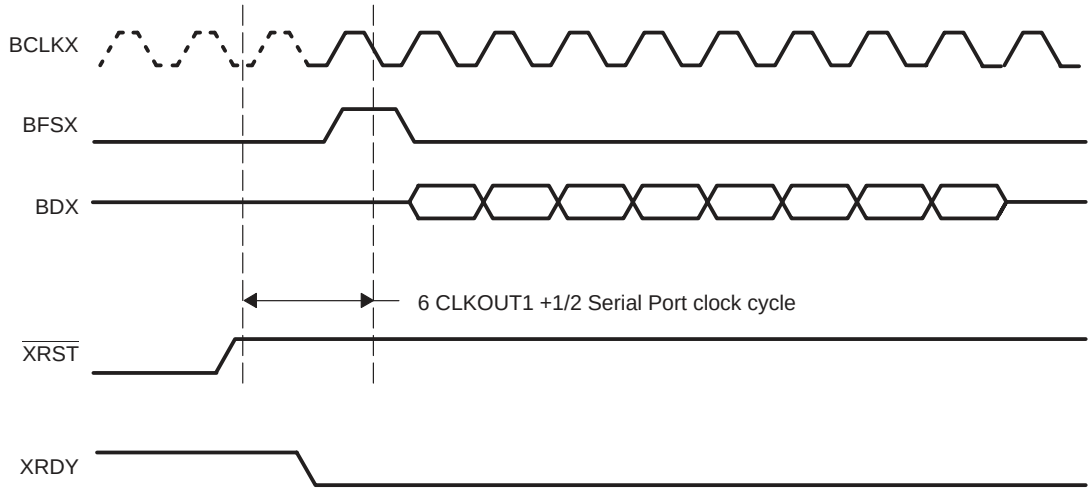
In autobuffering mode, for receive, and transmit with external frame sync (TXM = 1), the BSP must be taken out of reset at least six CLKOUT1 cycles plus 1/2 serial port clock cycle prior to the edge of the clock which detects the active frame sync pulse (whether the clock has been running previously or not) for proper operation. This is due to the time delay for the ABU logic to be activated. See Figure 9–40.

Transmit operations with internal clock and frame sync are not subject to this requirement since frame sync is internally generated automatically after $\overline{\text{XRST}}$ is cleared.

Note, however, that if external serial port clock is used with internal frame sync, and if the clock is not running when $\overline{\text{XRST}}$ is cleared, frame sync generation may be delayed depending on the timing of clearing $\overline{\text{XRST}}$ with respect to the clock.

Figure 9–40 illustrates autobuffering mode initialization timing requirements for the transmitter with external clock and frame sync. The figure shows standard mode operation with external frame (TXM = 0) and clock (MCM = 0), active high frame sync (FSP = 0), and data sampled on rising edge (CLKP = 0).

Figure 9–40. Autobuffering Mode Initialization Timing



9.8.3.2 Software Initialization Examples

In order to start or restart BSP operation in standard mode, the same steps are performed in software as with initializing the SP (see Section 9.7, *Serial Port Interface*, on page 9-23), in addition to which, the SPCE register must be initialized to configure any of the enhanced features desired. To start or restart the

BSP in autobuffering mode, a similar set of steps must also be performed, in addition to which, the autobuffering registers must be initialized.

The following two code examples illustrate initializing the serial port interface for autobuffering mode operation. In both cases, the code is written assuming that transmit and receive interrupts are used to service the ABU interrupts, however, polling of the interrupt flag register (IFR) could also be used. Both the transmit and receive sections can be initialized at the same time or separately depending upon system requirements.

Example 9–5 initializes the serial port for transmit operations only, with burst mode, external frame sync and external clock selected. The selected data format is 10 bits, with frame sync and clock polarities selected to be high true. Transmit autobuffering is enabled by setting the BXE bit in the ABUC section of SPCE, and HALTX has been set to 1, which causes transmission to halt when half of the defined buffer is transmitted.

Example 9–6 initializes the serial port for receive operations only, with continuous mode selected. Frame sync and clock polarities are selected to be low true, data format is 16 bits, and frame ignore is selected so that two received data bytes are packed into a single received word to minimize memory requirements. Receive autobuffering is enabled by setting the BRE bit in the ABUC section of SPCE.

Note that in Example 9–5 and Example 9–6, the transmit and receive interrupts used are those that the BSP occupies on the ’C56 and ’C57, the two main devices which include the BSP. However, on other devices which use the BSP, different interrupts may be used, therefore, appropriate device documentation should be consulted. Also, for both examples, it is assumed that DP has been initialized to 0 and that interrupts are disabled (INTM = 1) when entering the routines.

Example 9–5. Transmit Initialization in Burst Mode with External Frame Sync and External Clock (Format is 10 bits)

```
OPL    #00080h,IMR   ;enable transmit interrupt (XINT)
SPLK   #00008h,BSPC  ;configure serial port SPC register
                     ;(XRST=0)
SPLK   #01480h,SPCE  ;configure serial port SPCE register
SPLK   #XTOP,AXR     ;init address of buffer start in AXR
SPLK   #XSIZE,BKX    ;init size of buffer
OPL    #00080h,IFR   ;clear any latched transmit interrupt
OPL    #00040h,BSPC  ;start transmit part (XRST=1)
CLRC   INTM          ;enable interrupts
```

Example 9–6. Receive Initialization in Continuous Mode (Format is 16 bits)

```
OPL    #00040h,IMR   ;enable receive interrupt (RINT)
SPLK   #00000h,BSPC  ;reset and configure serial port SPC
                     ;(RRST=0)
SPLK   #02160h,SPCE  ;configure serial port SPCE register
SPLK   #RTOP,ARR     ;init pointer with top of buffer address
SPLK   #RSIZE,BKR    ;init size of receive buffer
OPL    #00040h,IFR   ;clear any latched receive interrupt
OPL    #0080h,BSPC   ;start receive part
CLRC   INTM          ;enable interrupts
```

9.8.4 BSP Operation in Power-Down Mode

The ’C5x offers several power down modes which allow part or all of the device to enter a dormant state and dissipate considerably less power than when running normally. Power down mode may be invoked in several ways, including either executing the IDLE/IDLE2 instructions or driving the $\overline{\text{HOLD}}$ input low with the HM status bit set to 1. The BSP, like other peripherals (timer, standard serial port), can take the CPU out of IDLE using the transmit interrupt (XINT) or receive interrupt (RINT).

When in IDLE or HOLD mode, the BSP continues to operate, as is the case with the SP. When in IDLE2, unlike the SP and other on-chip peripherals which are stopped with this power-down mode, the BSP can still be operated.

In standard mode, if the BSP is using external clock and frame sync while the device is in IDLE2, the port will continue to operate, and a transmit interrupt (XINT) or receive interrupt (RINT) will take the device out of IDLE2 mode if INTM = 0 before the device executes the IDLE2 instruction. With internal clock and/or frame sync, the BSP remains in IDLE2 until the CPU resumes operation.

In autobuffering mode, if the BSP is using external clock and frame sync while the device is in IDLE2, a transmit/receive event will cause the internal BSP clock to be turned on for the cycles required to perform the DXR (or DRR) to memory transfer. The internal BSP clock is then turned off automatically as soon as the transfer is complete so the device will remain in IDLE2 mode. The device is awakened from IDLE2 by the ABU transmit interrupt (XINT) or receive interrupt (RINT) when the transmit/receive buffer has been halfway or entirely emptied or filled if INTM = 0 before the device executes the IDLE2 instruction.

9.9 Time-Division Multiplexed (TDM) Serial Port Interface

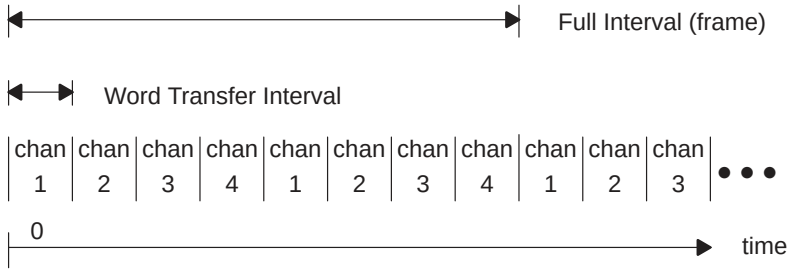
The time-division multiplexed (TDM) serial port allows the 'C5x device to communicate serially with up to seven other devices. The TDM port, therefore, provides a simple and efficient interface for multiprocessing applications.

The TDM serial port is a superset of the serial port described in Section 9.7 on page 9-23. By means of the TDM bit in the TDM serial port control register (TSPC), the port can be configured in multiprocessing mode (TDM = 1) or stand-alone mode (TDM = 0). When in stand-alone mode, the port operates as described in Section 9.7. When in multiprocessing mode, the port operates as described in this section. The port can be shut down for low power consumption via the $\overline{\text{XRST}}$ and $\overline{\text{RRST}}$ bits, as described in Section 9.7.

9.9.1 Basic Time-Division Multiplexed Operation

Time-division multiplexing is the division of time intervals into a number of subintervals, with each subinterval representing a communications channel according to a prespecified arrangement. Figure 9–41 shows a 4-channel TDM scheme. Note that the first time slot is labeled chan 1 (channel 1), the next chan 2 (channel 2), etc. Channel 1 is active during the first communications period and during every fourth period thereafter. The remaining three channels are interleaved in time with channel 1.

Figure 9–41. Time-Division Multiplexing



The 'C5x TDM port uses eight TDM channels. Which device is to transmit and which device or devices is/are to receive for each channel may be independently specified. This results in a high degree of flexibility in interprocessor communications.

9.9.2 TDM Serial Port Interface Registers

The TDM serial port operates through six memory-mapped registers and two other register (TRSR and TXSR) that are not directly accessible to the program, but are used in the implementation of the double-buffering capability. These eight registers are listed in Table 9–21.

Table 9–21. TDM Serial Port Registers

Address	Register	Description
0030h	TRCV	TDM data receive register
0031h	TDXR	TDM data transmit register
0032h	TSPC	TDM serial port control register
0033h	TCSR	TDM channel select register
0034h	TRTA	TDM receive/transmit address register
0035h	TRAD	TDM receive address register
—	TRSR	TDM data receive shift register
—	TXSR	TDM data transmit shift register

- TDM data receive register (TRCV). The 16-bit TDM data receive register (TRCV) holds the incoming TDM serial data. The TRCV has the same function as the DRR, described in Section 9.7 on page 9-23.

- TDM data transmit register (TDXR). The 16-bit TDM data transmit register (TDXR) holds the outgoing TDM serial data. The TDXR has the same function as the DXR, described in Section 9.7 on page 9-23.

- TDM serial port control register (TSPC). The 16-bit TDM serial port control register (TSPC) contains the mode control and status bits of the TDM serial port interface. The TSPC is identical to the SPC (Figure 9–14) except that bit 0 serves as the TDM mode enable control bit in the TSPC. The TDM bit configures the port in TDM mode (TDM = 1) or stand-alone mode (TDM = 0). In stand-alone mode, the port operates as a standard serial port as described in Section 9.7 on page 9-23.

- TDM channel select register (TCSR). The 16-bit TDM channel select register (TCSR) specifies in which time slot(s) each ’C5x device is to transmit.

- TDM receive/transmit address register (TRTA). The 16-bit TDM receive/transmit address register (TRTA) specifies in the eight LSBs (RA0–RA7) the receive address of the ’C5x device and in the eight MSBs (TA0–TA7) the transmit address of the ’C5x device.

- TDM receive address register (TRAD). The 16-bit TDM receive address register (TRAD) contains various information regarding the status of the TDM address line (TADD).

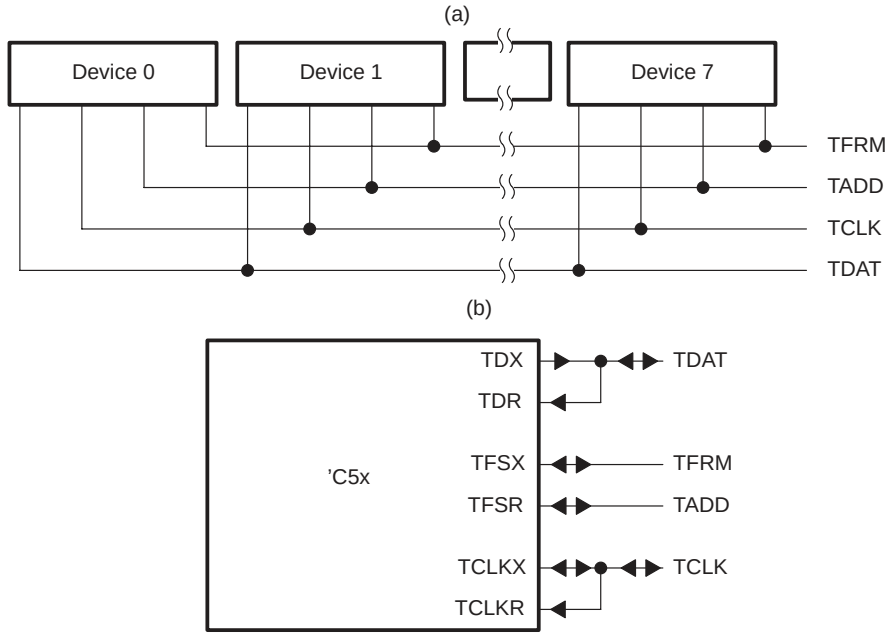
- TDM data receive shift register (TRSR). The 16-bit TDM data receive shift register (TRSR) controls the storing of the data, from the input pin, to the TRCV. The TRSR has the same function as the RSR, described in Section 9.7 on page 9-23.

- TDM data transmit shift register (TXSR). The 16-bit TDM data transmit shift register (TXSR) controls the transfer of the outgoing data from the TDXR and holds the data to be transmitted on the data-transmit (TDX) pin. The TXSR has the same function as the XSR, described in Section 9.7 on page 9-23.

9.9.3 TDM Serial Port Interface Operation

Figure 9–42(a) shows the ’C5x TDM port architecture. Up to eight devices can be placed on the four-wire serial bus. This four-wire bus consists of a conventional serial port’s bus of clock, frame, and data (TCLK, TFRM, and TDAT) wires plus an additional wire (TADD) that carries the device addressing information. Note that the TDAT and TADD signals are bidirectional signals and are often driven by different devices on the bus during different time slots within a given frame of operation.

Figure 9–42. TDM 4-Wire Bus



The TADD line, which is driven by a particular device for a particular time slot, determines which device(s) in the TDM configuration should execute a valid TDM receive during that time slot. This is similar to a valid serial port read operation, as described in Section 9.7, *Serial Port Interface*, on page 9-23, except that some corresponding TDM registers are named differently. The TDM receive register is TRCV, and the TDM receive shift register is TRSR. Data is transmitted on the bidirectional TDAT line.

Note that in Figure 9–42(b) the device TDX and TDR pins are tied together externally to form the TDAT line. Also, note that only one device can drive the data and address line (TDAT and TADD) in a particular slot. All other devices' TDAT and TADD outputs should be in the high-impedance state during that slot, which is accomplished through proper programming of the TDM port control registers (this is described in detail later in this section). Meanwhile, in that particular slot, all the devices (including the one driving that slot) sample the TDAT and TADD lines to determine if the current transmission represents valid data to be read by any one of the devices on the bus (this is also discussed in detail later in this section). When a device recognizes an address to which it is supposed to respond, a valid TDM read then occurs, the value is transferred from TRSR to the TRCV register. A receive interrupt (TRNT) is generated, which indicates that TRCV has valid receive data and can be read.

All TDM port operations are synchronized by the TCLK and TFRM signals. Each of them are generated by only one device (typically the same device), referred to as the TCLK and TFRM source(s). The word master is not used here because it implies that one device controls the other, which is not the case, and TCSR must be set to prevent slot contention. Consequently, the remaining devices in the TDM configuration use these signals as inputs. Figure 9–42(b) shows that TCLKX and TCLKR are externally tied together to form the TCLK line. Also, TFRM and TADD originate from the TFSX and TFSR pins respectively. This is done to make the TDM serial port also easy to use in standard mode.

TDM port operation is controlled by six memory-mapped registers. The layout of these registers is shown in Figure 9–43. The TRCV and TDXR registers have the same functions as the DRR and DXR registers respectively, described in Section 9.7, *Serial Port Interface*. The TSPC register is identical to the SPC register except that bit 0 serves as the TDM mode enable control bit in the TSPC. This bit configures the port in TDM mode (TDM=1) or stand-alone mode (TDM=0). In stand-alone mode, the port operates as a standard serial port as described in Section 9.7. Refer to Section 9.7, *Serial Port Interface*, on page 9-23 for additional information about the function of the bits in these registers.

Figure 9–43. TDM Serial Port Registers Diagram

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRCV	Receive Data															
TDXR	Transmit Data															
TSPC	Free	Soft	X	X	XRDY	RRDY	IN1	IN0	$\overline{\text{RRST}}$	$\overline{\text{XRST}}$	TXM	MCM	X	0	0	TDM
TCSR	X	X	X	X	X	X	X	X	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
TRTA	TA7	TA6	TA5	TA4	TA3	TA2	TA1	TA0	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
TRAD	X	X	X2	X1	X0	S2	S1	S0	A7	A6	A5	A4	A3	A2	A1	A0

Note: X=Don't care.

When TDM mode is selected, the DLB and FO bits in the TSPC are hard-configured to 0, resulting in no access to the digital loopback mode and in a fixed word length of 16 bits (a different type of loopback is discussed in the example in subsection 9.9.6 on page 9-82). Also, the value of FSM does not affect the port when TDM=1, and the states of the underflow and overrun flags are indeterminate (subsection 9.9.5, *TDM Serial Port Interface Exception Conditions*, on page 9-82 explains how exceptions are handled in TDM mode). If TDM=1, changes made to the contents of the TSPC become effective upon completion of channel 7 of the current frame. Thus the TSPC value cannot be changed for the current frame; any changes made will take effect in the next frame.

The source device for the TCLK and TFRM timing signals is set by the MCM and TXM bits, respectively. The TCLK source device is identified by setting the MCM bit of its TSPC register to 1. Typically, this device is the same one that supplies the TDM port clock signal TCLK. The TCLKX pin is configured as an input if MCM=0 and an output if MCM=1. In the latter case (internal 'C5x clock), the device whose MCM=1 supplies the clock (TCLK frequency=one fourth of CLKOUT1 frequency) for all devices on the TDM bus. The clock can be supplied by an external source if MCM=0 for all devices. TFRM can also be supplied externally if TXM=0. An external TFRM, however, must meet TDM receive timing specifications with respect to TCLK for proper operation. No more than one device should have MCM or TXM set to 1 at any given time. The specification of which device is to supply clock and framing signals is typically made only once, during system initialization.

The TDM channel select register (TCSR) of a given device specifies in which time slot(s) that device is to transmit. A 1 in any one or more of bits 0–7 of the TCSR sets the transmitter active during the corresponding time slot. Again, a key system-level constraint is that no more than one device can transmit

during the same time slot; devices do ***not*** check for bus contention, and slots must be consistently assigned. As in TSPC operation, a write to TCSR during a particular frame is valid only during the next frame. However, a given device can transmit in more than one slot. This is discussed in more detail in subsection 9.9.4, *TDM Mode Transmit and Receive Operations*, on page 9-80, with an emphasis on the utilization of TRTA, TDXR, and TCSR in this respect.

The TDM receive/transmit address register (TRTA) of a given device specifies two key pieces of information. The lower half specifies the receive address of the device, while the upper half of TRTA specifies the transmit address. The receive address (RA7–RA0, refer to Figure 9–43) is the 8-bit value that a device compares to the 8-bit value it samples on the TADD line in a particular slot to determine whether it should execute a valid TDM receive. The receive address, therefore, establishes the slots in which that device may receive, dependent on the addresses present in those slots, as specified by the transmitting devices. This process occurs on each device during every slot.

The transmit address (TA7–TA0, refer to Figure 9–43) is the address that the device drives on the TADD line during a transmit operation on an assigned slot. The transmit address establishes which receiving devices may execute a valid TDM receive on the driven data.

Only one device at a time can drive a transmit address on TADD. Each processor bit-wise-logically-ANDs the value it samples on the TADD line with its receive address (RA7–RA0). If this operation results in a nonzero value, then a valid TDM receive is executed on the processor(s) whose receive addresses match the transmitted address. Thus, for one device to transmit to another, there must be at least one bit in the upper half of the transmitting device's TRTA (the transmit address) with a value of 1 that matches one bit with a value of 1 in the lower half of TRTA (the receive address) of the receiving device. This method of configuration of TRTA allows one device to transmit to one or more devices, and for any one device to receive from one or more than one transmitter. This can also allow the transmitting device to control which devices receive, without the receive address on any of the devices having to be changed.

The TDM receive address register (TRAD) contains various information regarding the status of the TADD line which can be polled to verify the previous values of this signal and to verify the relationship between instruction cycles and TDM port timing.

Bits 13–11 (X2–X0) contain the current slot number value, regardless of whether a valid data receive was executed in that slot or not. This value is latched at the beginning of the slot and retained only until the end of the slot.

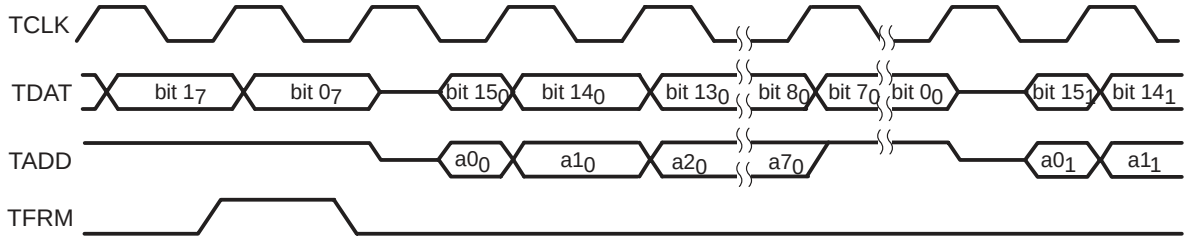
Bits 10–8 (S02–S0) hold the number of the last slot plus one (modulo eight) in which data was received (that is, if the last valid data read occurred in slot 5 in the previous frame, these bits would contain the number six). This value is latched during the TDM receive interrupt (TRNT) at the end of the slot in which the last valid data receive occurred, and maintained until the end of the next slot in which a valid receive occurs.

Bits 7–0 (A7–A0) hold the last address sampled on the TADD line, regardless of whether a valid data receive was executed or not. This value is latched halfway through each slot (so the value on the TADD may be shifted in) and maintained until halfway through the next slot, whether a valid receive is executed or not.

9.9.4 TDM Mode Transmit and Receive Operations

Figure 9–44 shows the timing for TDM port transfers. The TCLK and TFRM signals are generated by the timing source device. The TCLK frequency is one fourth the frequency of CLKOUT1 if generated by a 'C5x device. The TFRM pulse occurs every 128 TCLK cycles and is timed to coincide with bit 0 of slot 7, which is the last bit of the previous frame. The relationship of TFRM and TCLK allows 16 data bits for each of eight time slots to be driven on the TDAT line, which also permits the processor to execute a maximum of 64 instructions during each slot, assuming that a 'C5x internal clock is used. Beginning with slot 0 and with the MSB first, the transmitter drives 16 data bits for each slot, with each bit having a duration of one TCLK cycle, with the exception of the first bit of each slot, which lasts only half of one bit time. Note that data is both clocked onto the TDAT line by the transmitting device and sampled from the TDAT line by receiving devices on the rising edge of TCLK (see the data sheet for detailed TDM interface timings).

Figure 9–44. Serial Port Timing (TDM Mode)



Simultaneous with data transfer, the transmitting device also drives the TADD line with the transmit address for each slot. This information, unlike that on TDAT, is only one byte long and is transmitted with the LSB first for the first half of the slot. During the second half of the slot (that is, the last eight TCLK periods) the TADD line is driven high. The TDM receive logic samples the TADD line only for the first eight TCLK periods, ignoring it during the second half of the slot. Therefore, the transmitting device (if not a 'C5x) could drive TADD high or low during that time period. Note that, like TDAT, the first TADD bit transmitted lasts for only one half of one TCLK cycle.

If no device on the TDM bus is configured to transmit in a slot (that is, none of the devices has a 1 for the corresponding slot in their TCSR register), that slot is considered empty. In an empty slot, both TADD and TDAT are high impedance. This condition has the potential for spurious receives, however, because TDAT and TADD are always sampled, and a device performs a valid TDM reception if its receive address matches the address on the TADD line. To avoid spurious reads, a 1-kilohm pull-down resistor *must* be tied to the TADD line. This causes the TADD line to read low on empty slots. Otherwise, any noise on the TADD line that happens to match a particular receive address would result in a spurious read. If power dissipation is a concern and the resistor is not desired, then an arbitrary processor with transmit address equal to 0h can drive empty slots by writing to TDXR in those slots. Slot manipulation is explained later in this section. The 1-kilohm resistor is not required on the TDAT line.

An empty TDM slot can result in the following cases: the first obvious case, as mentioned above, occurs when no device has its TCSR configured to transmit in that slot. A second more subtle case occurs when TDXR has not been loaded before a transmit slot in a particular frame. This may also happen when the TCSR contents are changed, since the actual TCSR contents are not updated until the next TFRM pulse occurs. Therefore, any subsequent change takes effect only in the next frame. The same is true for the receive address (the lower half of TRTA). The transmit address (upper half of TRTA), however, and TDXR, clearly, may be changed within the current frame for a particular slot, assuming that the slot has not yet been reached when the instruction to load the TRTA or TDXR is executed. Note that it is not necessary to load the transmit address each time TDXR is loaded; when a TDXR load occurs and a transmission begins, the current transmit address in TRTA is transmitted on TADD.

The current slot number may be obtained by reading the X2–X0 bits in TRAD. This affords the flexibility of reconfiguring the TDM port on a slot-by-slot basis, and even slot sharing if desired. The key to utilizing this capability is to understand the timing relationship between the instructions being executed and the

frame/slots of the TDM port. If the TDM port is to be manipulated on a slot-by-slot basis, changes must be made to appropriate registers quickly enough for the desired effect to take place at the desired time. It is also important to take into account that the TCSR and the receive address (lower half of TRTA) take effect only at the start of a new frame, while the transmit address (upper half of TRTA) and TDXR (transmit data) can take effect at the start of a new slot, as mentioned previously.

Note that if the transmit address is being changed on the fly, care should be exercised not to corrupt the receive address, since both addresses are located in the TRTA register, thus maintaining the convention of allowing the transmitting device to specify which devices can receive.

9.9.5 TDM Serial Port Interface Exception Conditions

Because of the nature of the TDM architecture, with the ability for one processor to transmit in multiple slots, the concepts of overrun and underflow become indeterminate. Therefore, the overrun and underflow flags are not active in TDM mode.

In the receiver, if TRCV has not been read and a valid receive operation is initiated (because of the value on TADD and the device's receive address), the present value of TRCV is overwritten; the receiver is *not* halted. On the other hand, if TDXR has not been updated before a transmission, the TADD or TDAT lines are not driven, and these pins remain in the high-impedance state. This mode of operation prevents spurious transmits from occurring.

If a TFRM pulse occurs at an improper time during a frame, the TDM port is not able to continue functioning properly, since slot and bit numbers become ambiguous when this occurs. Only one TFRM should occur every 128 TCLK cycles. Unlike the serial port, the TDM port cannot be reinitialized with a frame sync pulse during transmission. To correct an improperly timed TFRM pulse, the TDM port must be reset.

9.9.6 Examples of TDM Serial Port Interface Operation

The following is an example of TDM serial port operation, showing the contents of some of the key device registers involved, and explaining the effect of this configuration on port operation. In this example, eight devices are connected to the TDM serial port as shown in Figure 9–42 on page 9-76.

Table 9–22 shows the TADD value during each of the eight channels given the transmitter and receiver designations shown. This example shows the configuration for eight devices to communicate with each other. In this example, device 0 broadcasts to all other device addresses during slot 0. In subsequent frames, devices 1–7 each communicate to one other processor.

Table 9–22. Interprocessor Communications Scenario

Channel	TADD Data	Transmitter Device	Receiver Device(s)
0	0FEh	0	1–7
1	40h	7	6
2	20h	6	5
3	10h	5	4
4	08h	4	3
5	04h	3	2
6	02h	2	1
7	01h	1	0

Table 9–23 shows the TDM serial port register contents of each device that results in the scenario given in Table 9–22. Device 0 provides the clock and frame control signals for all channels and devices. The TCSR and TRTA contents specify which device is to transmit on a given channel and which devices are to receive.

Table 9–23. TDM Register Contents

Device	TSPC	TRTA	TCSR
0	xxF9h	0FE01h	xx01h
1	xxC9h	0102h	xx80h
2	xxC9h	0204h	xx40h
3	xxC9h	0408h	xx20h
4	xxC9h	0810h	xx10h
5	xxC9h	1020h	xx08h
6	xxC9h	2040h	xx04h
7	xxC9h	4080h	xx02h

In this example, the transmit address of a given device (the upper byte of TRTA) matches the receive address (the lower byte of TRTA) of the receiving device. Note, however, that it is not necessary for the transmit and receive addresses to match exactly; the matching operation implemented in the receiver is a bitwise AND operation. Thus, it is only necessary that one bit in the field matches for a receive to occur. The advantage of this scheme is that a transmitting device can select the device or devices to receive its transmitted data by simply changing its transmit address (as long as each devices' receive address is unique, the receive address of the receiving device does not need to be changed). In the example, device 0 can transmit to any combination of the other devices by merely writing to the upper byte of TRTA. Therefore, if a transmitting device changed its TRTA to 8001h on the fly, it would transmit only to device 7.

A device may also transmit to itself, because both the transmit and receive operations are executed on the rising edge of TCLK (see the data sheet for detailed TDM interface timings). To enable this type of loopback, it is necessary to use the standard TDM port interface connections as shown in Figure 9–42. Then, if device 0 has a TRTA of 0101h, it would transmit only to itself.

Another example of TDM port operation is provided in the code sequence of Example 9–7 in which a one-way transmit of a sequence of values from device 0 to device 1 is shown. The values are stored in each device in a block from 4000h to 6000h in data memory. Device 0 transmits in slot 0 and has a transmit address of 01h. It waits in a $\overline{\text{BIO}}$ loop for a ready-to-receive signal (XF) from device 1, and initializes the transfer data with a value of 0. Only its transmit interrupt is enabled and its transmit ISR writes the value it will send into its own memory.

Example 9–7. Device 0 Transmit Code (TDM Operation)

```
* Device 0 – Transmit side
                :
                :
                :
        SPLK #1h, TCSR          ;Setup TCSR to xmt on
                                ;slot 0
        SPLK #100h, TRTA        ;Setup transmit address

                                ;Set up TSPC as TCLK, TFRM
                                ;source
        SPLK #0039h, TSPC       ;Set TXM=MCM=FSM=TDM=1,
                                ;DLB=FO=0.
                                ;And put TDM into reset
                                ;(XRST=RRST=0)
        SPLK #00F9h, TSPC       ;Take TDM out of reset

                                ;Setup interrupts
        SPLK #0ffffh, IFR       ;clear IFR
        SPLK #080h, IMR         ;Turn on TXNT

        CLRC INTM               ;enable interrupts

TILOOP  BCND TSENDZ, BIO        ;Wait for ready–to–
        B    TILOOP             ;receive from other device

TSENDZ  LACL #0                 ;First transmission/write
                                ;value is 0.
        LAR  AR7, #4000h        ;Setup where to write
        SACL *                  ;Write first value
        SACL TDXR               ;Transmit first value

SELF2   B    SELF2              ;Wait for interrupts

_ISR
        LACC AR7                ;Check if past 0x6000
        SUB  #6000h             ;i.e. end of block
        BCND END_TDMP, GEQ      ;Go to tight loop if so.

                                ;Add one and transmit
        LACL *+                 ;Load value
        ADD  #1                 ;Add one
        SACL *                  ;Write value
        SACL TDXR               ;Transmit value
        RETE

END_TDMP   B  END_TDMP          ;Sit in tight loop after
                                ;block is complete.
                  :
                  :
                  :
```

Example 9–8 shows the code in device 1. It has a receive address of 01h and sends a ready-to-receive signal (XF) to device 0. Only its receive interrupt is enabled, and its receive ISR reads from the TRCV, writes to the block, and then checks to see if it has reached the end of the block.

Example 9–8. Device 1 Receive Code (TDM Operation)

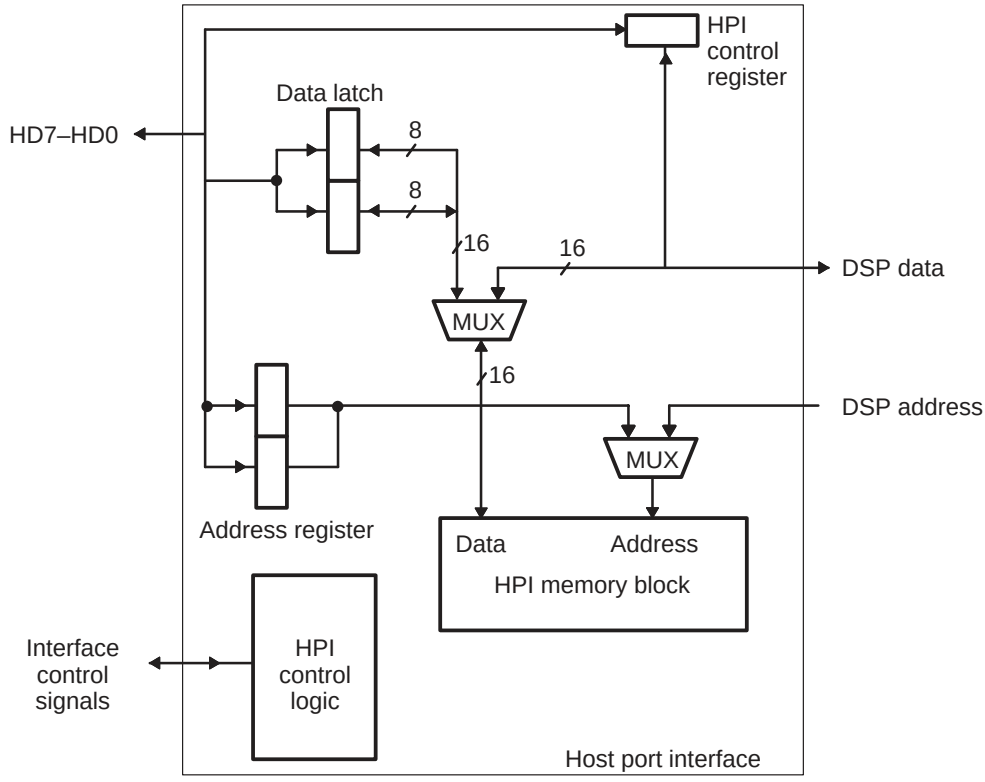
```
*Device 1 – receive side
        SPLK #0h, TCSR          ; Setup TCSR to xmt on
                                ; no slots
        SPLK #001h, TRTA        ; Setup receive address

                                ; Set TDM as TCLK, TFRM
                                ; receive
        SPLK #0009h, TSPC       ; Set TXM=MCM=DLB=FO=0,
                                ; FSM=TDM=1.
                                ; And put TDM into reset
                                ; (XRST=RRST=0)
        SPLK #00C9h, TSPC       ; Take TDM out of reset

                                ; Setup interrupts
        SPLK #0ffffh, IFR       ; clear IFR
        SPLK #040h, IMR         ; Mask on TRNT

        CLRC INTM               ; enable interrupts
        LAR  AR7, #4000h        ; Setup where to write
                                ; received data
        CLRC XF                 ; Signal ready to receive

SELF2   B    SELF2              ; Wait for interrupts

_ISR
        LACC TRCV               ; Load received value
        SACL *+                 ; Write to memory block
        LACC AR7                ; Check if past 0x6000
        SUB  #6000h             ; i.e. end of block
        BCND END_TDMP, GEQ      ; Go to tight loop if so
        RETE

END_TDMP   B  END_TDMP          ; Sit in tight loop after
                                ; block is complete.
```

9.10 Host Port Interface

The host port interface (HPI) is an 8-bit parallel port used to interface a host device or host processor to the ’C5x. Information is exchanged between the ’C5x and the host device through on-chip ’C5x memory that is accessible by both the host and the ’C5x. The HPI is available on the ’LC57 and ’C57S devices.

The HPI is designed to interface to the host device as a peripheral, with the host device as master of the interface, therefore greatly facilitating ease of access by the host. The host device communicates with the HPI through dedicated address and data registers, to which the ’C5x does not have direct access, and the HPI control register, using the external data and interface control signals (see Figure 9–45). Both the host device and the ’C5x have access to the HPI control register.

Figure 9–45. Host Port Interface Block Diagram



The HPI provides 16-bit data to the ’C5x while maintaining the economical 8-bit external interface by automatically combining successive bytes transferred into 16-bit words. When the host device performs a data transfer with the HPI registers, the HPI control logic automatically performs an access to a dedicated 2K-word block of internal ’C5x single access RAM to complete the transaction. The ’C5x can then access the data within its memory space. The HPI RAM can also be used as general purpose single access data or program RAM.

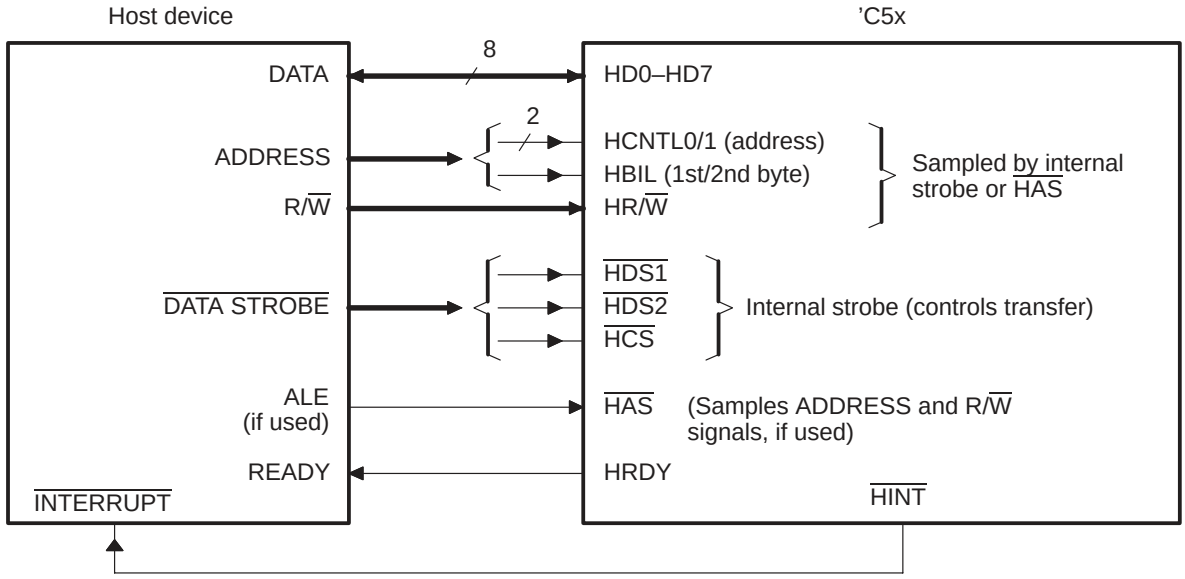
The HPI has two modes of operation, shared-access mode (SAM) and host-only mode (HOM). In shared-access mode, the normal mode of operation, both the ’C5x and the host can access HPI memory. In this mode, asynchronous host accesses are resynchronized internally and, in the case of a conflict between a ’C5x and a host cycle, the host has access priority and the ’C5x waits one cycle. In host-only mode, only the host can access HPI memory while the ’C5x is in reset or in IDLE2 with all internal and even external clocks stopped. The host can therefore access the HPI RAM while the ’C5x is in its minimum power consumption configuration.

The HPI supports high speed, back-to-back host accesses. In shared-access mode, the HPI can transfer one byte every five CLKOUT1 cycles (that is 64M bps) with the ’C5x running at a 40-MHz CLKOUT1. The HPI is designed so the host can take advantage of this high bandwidth and run at frequencies up to (Fd*n)/5, where Fd is the ’C5x CLKOUT1 frequency and n is the number of host cycles for an external access. Therefore, with a 40-MHz ’C5x and common values of 4 (or 3) for n, the host can run at speeds of up to 32 (or 24) MHz without requiring wait states. In the host-only mode, the HPI supports even higher speed back-to-back host accesses on the order of one byte every 50 ns (that is, 160M bps), independent of the ’C5x clock rate (refer to the TMS320C5x data sheet for specific detailed timing information).

9.10.1 Basic Host Port Interface Functional Description

The external HPI interface consists of the 8-bit HPI data bus and control signals that configure and control the interface. The interface can connect to a variety of host devices with little or no additional logic necessary. Figure 9–46 shows a simplified diagram of a connection between the HPI and a host device.

Figure 9–46. Generic System Block Diagram



The 8-bit data bus (HD0–HD7) exchanges information with the host. Because of the 16-bit word structure of the 'C5x, all transfers with a host must consist of two consecutive bytes. The dedicated HBIL pin indicates whether the first or second byte is being transferred. An internal control register bit determines whether the first or second byte is placed into the most significant byte of a 16-bit word. The host must not break the first byte/second byte (HBIL low/high) sequence of an ongoing HPI access. If this sequence is broken, data can be lost, and unpredictable operation can result.

The two control inputs (HCNTL0 and HCNTL1) indicate which internal HPI register is being accessed and the type of access to the register. These inputs, along with HBIL, are commonly driven by host address bus bits or a function of these bits. Using the HCNTL0/1 inputs, the host can specify an access to the HPI control (HPIC) register, the HPI address (HPIA) register (which serves as the pointer into HPI memory), or HPI data (HPID) register. The HPID register can also be accessed with an optional automatic address increment.

The autoincrement feature provides a convenient way of reading or writing to subsequent word locations. In autoincrement mode, a data read causes a postincrement of the HPIA, and a data write causes a preincrement of the HPIA. By writing to the HPIC, the host can interrupt the 'C5x CPU, and the $\overline{\text{HINT}}$ output can be used by the 'C5x to interrupt the host. The host can also acknowledge and clear $\overline{\text{HINT}}$ by writing to the HPIC.

Table 9–24 summarizes the three registers that the HPI utilizes for communication between the host device and the ’C5x CPU and their functions.

Table 9–24. HPI Registers Description

Name	Address	Description
HPIA	—	HPI address register. Directly accessible only by the host. Contains the address in the HPI memory at which the current access occurs.
HPIC	0500h	HPI control register. Directly accessible by either the host or by the ’C5x. Contains control and status bits for HPI operations.
HPID	—	HPI data register. Directly accessible only by the host. Contains the data that was read from the HPI memory if the current access is a read, or the data that will be written to HPI memory if the current access is a write.

The two data strobes ($\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$), the read/write strobe (HR/$\overline{\text{W}}$), and the address strobe ($\overline{\text{HAS}}$) enable the HPI to interface to a variety of industry-standard host devices with little or no additional logic required. The HPI is easily interfaced to hosts with multiplexed address/data bus, separate address and data buses, one data strobe and a read/write strobe, or two separate strobes for read and write. This is described in detail later in this section.

The HPI ready pin (HRDY) allows insertion of wait states for hosts that support a ready input to allow deferred completion of access cycles and have faster cycle times than the HPI can accept due to ’C5x operating clock rates. If HRDY, when used directly from the ’C5x, does not meet host timing requirements, the signal can be resynchronized using external logic if necessary. HRDY is useful when the ’C5x operating frequency is variable, or when the host is capable of accessing at a faster rate than the maximum shared-access mode access rate (up to the host-only mode maximum access rate). In both cases, the HRDY pin provides a convenient way to automatically (no software handshake needed) adjust the host access rate to a faster ’C5x clock rate or switch the HPI mode.

All of these features combined allow the HPI to provide a flexible and efficient interface to a wide variety of industry-standard host devices. Also, the simplicity of the HPI interface greatly simplifies data transfers both from the host and the ’C5x sides of the interface. Once the interface is configured, data transfers are made with a minimum of overhead at a maximum speed.

9.10.2 Details of Host Port Interface Operation

This subsection includes a detailed description of each HPI external interface pin function, as well as descriptions of the register and control bit functions. Logical interface timings and initialization and read/write sequences are discussed in subsection 9.10.3, *Host Read/Write Access to HPI*, on page 9-97.

The external HPI interface signals implement a flexible interface to a variety of types of host devices. Devices with single or multiple data strobes and with or without address latch enable (ALE) signals can easily be connected to the HPI.

Table 9–25 gives a detailed description of the function of each of the HPI external interface pins.

Table 9–25. HPI Signal Names and Functions

HPI Pin	Host Pin	State†	Signal Function
$\overline{\text{HAS}}$	Address latch enable (ALE) or Address strobe or unused (tied high)	I	Address strobe input. Hosts with a multiplexed address and data bus connect $\overline{\text{HAS}}$ to their ALE pin or equivalent. HBIL, HCNTL0/1, and HR/$\overline{\text{W}}$ are then latched on $\overline{\text{HAS}}$ falling edge. When used, $\overline{\text{HAS}}$ must precede the later of $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, or $\overline{\text{HDS2}}$ (see 'C5x data sheet for detailed HPI timing specifications). Hosts with separate address and data bus can connect $\overline{\text{HAS}}$ to a logic-1 level. In this case, HBIL, HCNTL0/1, and HR/$\overline{\text{W}}$ are latched by the later of $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$, or $\overline{\text{HCS}}$ falling edge while $\overline{\text{HAS}}$ stays inactive (high).
HBIL	Address or control lines	I	Byte identification input. Identifies first or second byte of transfer (but not most significant or least significant — this is specified by the BOB bit in the HPIC register, described later in this section). HBIL is low for the first byte and high for the second byte.
HCNTL0, HCNTL1	Address or control lines	I	Host control inputs. Selects a host access to the HPIA register, the HPI data latches (with optional address increment), or the HPIC register.
$\overline{\text{HCS}}$	Address or control lines	I	Chip select. Serves as the enable input for the HPI and must be low during an access but may stay low between accesses. $\overline{\text{HCS}}$ normally precedes $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$, but this signal also samples HCNTL0/1, HR/$\overline{\text{W}}$, and HBIL if $\overline{\text{HAS}}$ is not used and $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$ are already low (this is explained in further detail later in this subsection). Figure 9–47 on page 9-93 shows the equivalent circuit of the $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ inputs.

† I: Input
O: Output
Z: High impedance

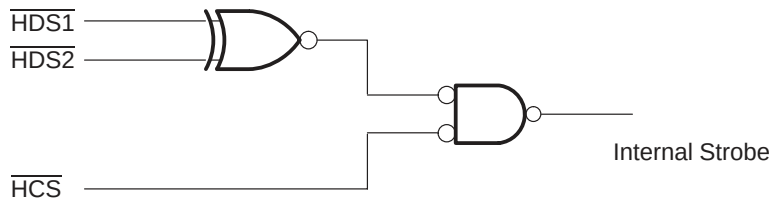
Table 9–25. HPI Signal Names and Functions (Continued)

HPI Pin	Host Pin	State†	Signal Function
HD7–HD0	Data bus	I/O/Z	Parallel bidirectional 3-state data bus. HD7 (MSB) through HD0 (LSB) are placed in the high-impedance state when not outputting (HDSx and HCS = 1) or when EMU1/OFF is active (low).
HDS1, HDS2	Read strobe and write strobe or data strobe	I	Data strobe inputs. Control transfer of data during host access cycles. Also, when HAS is not used, used to sample HBIL, HCNTL0/1, and HR/W when HCS is already low (which is the case in normal operation). Hosts with separate read and write strobes connect those strobes to either HDS1 or HDS2. Hosts with a single data strobe connect it to either HDS1 or HDS2, connecting the unused pin high. Regardless of HDS connections, HR/W is still required to determine direction of transfer. Because HDS1 and HDS2 are internally exclusive-NORed, hosts with a high true data strobe can connect this to one of the HDS inputs with the other HDS input connected low. Figure 9–47 on page 9-93 shows the equivalent circuit of the HDS1, HDS2, and HCS inputs.
HINT	Host interrupt input	O/Z	Host interrupt output. Controlled by the HINT bit in the HPIC. Driven high when the ’C5x is being reset. Placed in high impedance when EMU1/OFF is active (low).
HRDY	Asynchronous ready	O/Z	HPI ready output. When high, indicates that the HPI is ready for a transfer to be performed. When low, indicates that the HPI is busy completing the internal portion of the previous transaction. Placed in high impedance when EMU1/OFF is active (low). HCS enables HRDY; that is, HRDY is always high when HCS is high.
HR/W	Read/Write strobe, address line, or multiplexed address/data	I	Read/write input. Hosts must drive HR/W high to read HPI and low to write HPI. Hosts without a read/write strobe can use an address line for this function.

† I: Input
O: Output
Z: High impedance

The $\overline{HCS}$ input serves primarily as the enable input for the HPI, and the $\overline{HDS1}$ and $\overline{HDS2}$ signals control the HPI data transfer; however, the logic with which these inputs are implemented allows their functions to be interchanged if desired. If $\overline{HCS}$ is used in place of $\overline{HDS1}$ and $\overline{HDS2}$ to control HPI access cycles, HRDY operation is affected (since $\overline{HCS}$ enables HRDY and HRDY is always high when $\overline{HCS}$ is high). The equivalent circuit for these inputs is shown in Figure 9–47. The figure shows that the internal strobe signal that samples the HCNTL0/1, HBIL, and HR/$\overline{W}$ inputs (when $\overline{HAS}$ is not used) is derived from all three of the input signals, as the logic illustrates. Therefore, the latest of $\overline{HDS1}$, $\overline{HDS2}$, or $\overline{HCS}$ is the one which actually controls sampling of the HCNTL0/1, HBIL, and HR/$\overline{W}$ inputs. Because $\overline{HDS1}$ and $\overline{HDS2}$ are exclusive-NORed, both these inputs being low does not constitute an enabled condition.

Figure 9–47. Select Input Logic



When using the $\overline{HAS}$ input to sample HCNTL0/1, HBIL and HR/$\overline{W}$, this allows these signals to be removed earlier in an access cycle, therefore allowing more time to switch bus states from address to data information, facilitating interface to multiplexed address and data type buses. In this type of system, an ALE signal is often provided and would normally be the signal connected to $\overline{HAS}$.

The two control pins (HCNTL0 and HCNTL1) indicate which internal HPI register is being accessed and the type of access to the register. The states of these two pins select access to the HPI address (HPIA), HPI data (HPID), or HPI control (HPIC) registers. The HPIA register serves as the pointer into HPI memory, the HPIC contains control and status bits for the transfers, and the HPID contains the actual data transferred. Additionally, the HPID register can be accessed with an optional automatic address increment. Table 9–26 describes the HCNTL0/1 bit functions.

Table 9–26. HPI Input Control Signals Function Selection Descriptions

HCNTL1	HCNTL0	Description
0	0	Host can read or write the HPI control register, HPIC.
0	1	Host can read or write the HPI data latches. HPIA is automatically postincremented each time a read is performed and preincremented each time a write is performed.
1	0	Host can read or write the address register, HPIA. This register points to the HPI memory.
1	1	Host can read or write the HPI data latches. HPIA is not affected.

On the ’C57, HPI memory is a 2K × 16-bit word block of single-access RAM that can be configured to reside either from 1000h to 17FFh in data memory space or from 8800h to 8FFFh in program memory space. As with all single-access RAM blocks, the HPI RAM is affected by the ROM protection feature, if it is enabled. Also, the HPI memory may be located at different addresses on other ’C5x devices; consult the specific product documentation.

From the host interface, the 2K-word block of HPI memory can conveniently be accessed at addresses 0 through 7FFh; however, the memory can also be accessed by the host starting with any HPIA values with the 11 LSB’s equal to 0. For example, the first word of the HPI memory block, addressed at 1000h by the ’C57 in data memory space, can be accessed by the host with any of the following HPIA values: 0000h, 0800h,1000h,1800h, ··· F800h.

The HPI autoincrement feature provides a convenient way of accessing consecutive word locations in HPI memory. In the autoincrement mode, a data read causes a postincrement of the HPIA, and a data write causes a preincrement of the HPIA. Therefore, if a write is to be made to the first word of HPI memory with the increment option, due to the preincrement nature of the write operation, the HPIA should first be loaded with any of the following values: 07FFh, 0FFFh, 17FFh, ··· FFFFh. The HPIA is a 16-bit register and all 16 bits can be written to or read from, although with a 2K-word HPI memory implementation, only the 11 LSB’s of the HPIA are required to address the HPI memory. The HPIA increment and decrement affect all 16 bits of this register.

HPI Control Register Bits and Function

Four bits control HPI operation. These bits are BOB (which selects first or second byte as most significant), SMOD (which selects host or shared-access mode), and DSPINT and HINT (which can be used to generate ’C5x and host interrupts, respectively) and are located in the HPI control register (HPIC). A detailed description of the HPIC bit functions is presented in Table 9–27.

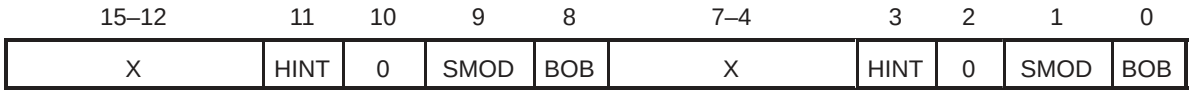
Table 9–27. HPI Control Register (HPIC) Bit Descriptions

Bit	Host Access	'C5x Access	Description
BOB	Read/Write	—	If BOB = 1, first byte is least significant. If BOB = 0, first byte is most significant. BOB affects both data and address transfers. Only the host can modify this bit and it is not visible to the 'C5x. BOB must be initialized before the first data or address register access.
DSPINT	Write	—	The host processor-to-'C5x interrupt. This bit can be written only by the host and is not readable by the host or the 'C5x. When the host writes a 1 to this bit, an interrupt is generated to the 'C5x. Writing a 0 to this bit has no effect. Always read as 0. When the host writes to HPIC, both bytes must write the same value. See this subsection for a detailed description of DSPINT function.
HINT	Read/Write	Read/Write	This bit determines the state of the 'C5x $\overline{\text{HINT}}$ output, which can be used to generate an interrupt to the host. HINT = 0 upon reset, which causes the external $\overline{\text{HINT}}$ output to be inactive (high). The HINT bit can be set only by the 'C5x and can be cleared only by the host. The 'C5x writes a 1 to HINT, causing the $\overline{\text{HINT}}$ pin to go low. The HINT bit is read by the host or the 'C5x as a 0 when the external $\overline{\text{HINT}}$ pin is inactive (high) and as a 1 when the $\overline{\text{HINT}}$ pin is active (low). For the host to clear the interrupt, however, it must write a 1 to HINT. Writing a 0 to the HINT bit by either the host or the 'C5x has no effect. See this subsection for a detailed description of HINT function.
SMOD	Read	Read/Write	If SMOD = 1, shared-access mode (SAM) is enabled: the HPI memory can be accessed by the 'C5x. If SMOD = 0, host-only mode (HOM) is enabled: the 'C5x is denied access to the entire HPI RAM block. SMOD = 0 during reset; SMOD = 1 after reset. SMOD can be modified only by the 'C5x but can be read by both the 'C5x and the host.

Because the host interface always performs transfers with 8-bit bytes and the control register is normally the first register accessed to set configuration bits and initialize the interface, the HPIC is organized on the host side as a 16-bit register with the same high and low byte contents (although access to certain bits is limited, as described previously) and with the upper bits unused on the 'C5x side. The control/status bits are located in the least significant four bits. The host accesses the HPIC register with the appropriate selection of HCNTL0/1, as described previously, and two consecutive byte accesses to the 8-bit HPI data bus. When the host writes to HPIC, both the first and second byte written must be the same value. The 'C57 accesses the HPIC at 500h in data memory space.

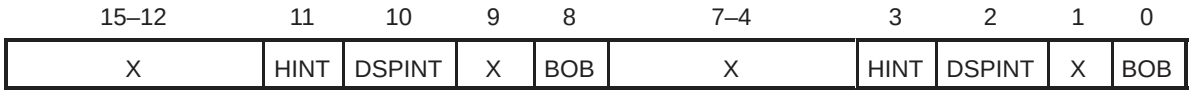
The layout of the HPIC bits is shown in Figure 9–48 through Figure 9–51. In the figures for read operations, if 0 is specified, this value is always read; if X is specified, an unknown value is read. For write operations, if X is specified, any value can be written. On a host write, both bytes must be identical. Note that bits 4–7 and 12–15 on the host side and bits 4–15 on the ’C5x side are reserved for future expansion.

Figure 9–48. HPIC Diagram — Host Reads from HPIC



Note: X = Unknown value is read.

Figure 9–49. HPIC Diagram — Host Writes to HPIC



Note: X = Any value can be written.

Figure 9–50. HPIC Diagram — ’C5x Reads From HPIC

Note: X = Unknown value is read.

Figure 9–51. HPIC Diagram — ’C5x Writes to HPIC

Note: X = Any value can be written.

Because the ’C5x can write to the SMOD and HINT bits, and these bits are read twice on the host interface side, the first and second byte reads by the host may yield different data if the ’C5x changes the state of one or both of these bits in between the two read operations. The characteristics of host and ’C5x HPIC read/write cycles are summarized in Table 9–28.

Table 9–28. HPIC Host/’C5x Read/Write Characteristics

Device	Read	Write
Host	2 bytes	2 bytes (Both bytes must be equal)
’C5x	16 bits	16 bits

9.10.3 Host Read/Write Access to HPI

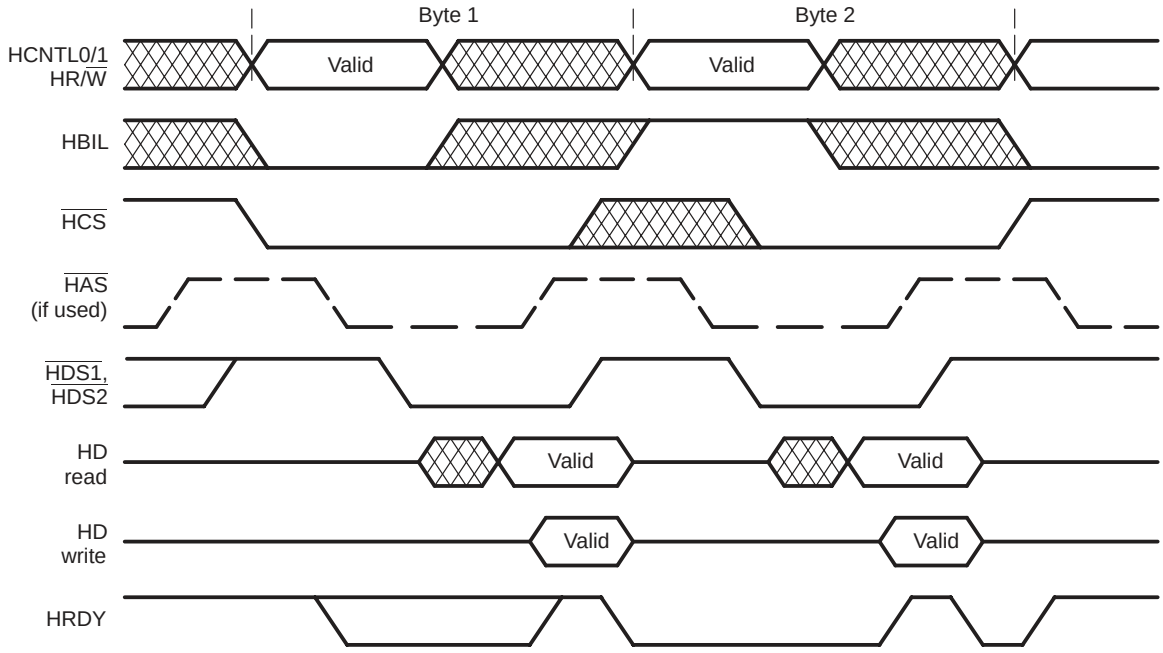
The host begins HPI accesses by performing the external interface portion of the cycle; that is, initializing first the HPIC register, then the HPIA register, and then writing data to or reading data from the HPID register. Writing to HPIA or HPID initiates an internal cycle that transfers the desired data between the HPID and the dedicated internal HPI memory. Because this process requires several ’C5x cycles, each time an HPI access is made, data written to the HPID is not written to the HPI memory until after the host access cycle, and the data read from the HPID is the data from the previous cycle. Therefore, when reading, the data obtained is the data from the location specified in the previous access, and the current access serves as the initiation of the next cycle. A similar sequence occurs for a write operation: the data written to HPID is not written to HPI memory until after the external cycle is completed. If an HPID read operation immediately follows an HPID write operation, the same data (the data written) is read.

The autoincrement feature available for HPIA results in sequential accesses to HPI memory by the host being extremely efficient. During random (nonsequential) transfers or sequential accesses with a significant amount of time between them, it is possible that the ’C5x may have changed the contents of the location being accessed between a host read and the previous host data read/write or HPIA write access, because of the prefetch nature of internal HPI operation. If this occurs, data different from the current memory contents may be read. Therefore, in cases where this is of concern in a system, two reads from the same address or an address write prior to the read access can be made to ensure that the most recent data is read.

When the host performs an external access to the HPI, there are two distinctly different types of cycles that can occur: those for which wait states are generated (the HRDY signal is active) and those without wait states. In general, when in shared-access mode (SAM), the HRDY signal is used; when in host-only mode (HOM), HRDY is not active and remains high; however, there are exceptions to this, which will be discussed.

For accesses utilizing the HRDY signal, during the time when the internal portion of the transfer is being performed (either for a read or a write), HRDY is low, indicating that another transfer cannot yet be initiated. Once the internal cycle is completed and another external cycle can begin, HRDY is driven high by the HPI. This occurs after a fixed delay following a cycle initiation (refer to the ’C5x data sheet for detailed timing information for HPI external interface timings). Therefore, unless back-to-back cycles are being performed, HRDY is normally high when the first byte of a cycle is transferred. The external HPI cycle using HRDY is shown in the timing diagram in Figure 9–52.

Figure 9–52. HPI Timing Diagram



In a typical external access, as shown in Figure 9–52, the cycle begins with the host driving HCNTL0/1, HR/$\overline{W}$, HBIL and $\overline{HCS}$, indicating specifically what type of transfer is to occur and whether the cycle is to be read or a write. Then the host asserts the $\overline{HAS}$ signal (if used) followed by one of the data strobe signals. If HRDY is not already high, it goes high when the previous internal cycle is complete, allowing data to be transferred, and the control signals are deasserted. Following the external HPI cycle, HRDY goes low and stays low for a period of approximately five CLKOUT1 cycles (refer to ’C5x data sheet for HPI timing information) while the ’C5x completes the internal HPI memory access, and then HRDY is driven high again. Note, however, HRDY is always high when $\overline{HCS}$ is high.

As mentioned previously, SAM accesses generally utilize the HRDY signal. The exception to the HRDY-based interface timings when in SAM occurs when reading HPIC or HPIA or writing to HPIC (except when writing 1 to either DSPINT or HINT). In these cases, HRDY stays high; for all other SAM accesses, HRDY is active.

Host access cycles when in HOM have timings different from the SAM timings described previously. In HOM, the CPU is not involved (with one exception), and the access can be completed after a short, fixed delay time. The exception to this occurs when writing 1s to the DSPINT or HINT bits in HPIC. In this case, the host access takes several CPU clock cycles, and SAM timings apply. Besides the HRDY timings and a faster cycle time, HOM access cycles are logically the same as SAM access cycles. A summary of the conditions under which the HRDY signal is active (where SAM timings apply) for host accesses is shown in Table 9–29. When HRDY is not active (HRDY stays high), HOM timings apply. Refer to the ’C5x data sheet for detailed HPI timing specifications.

Table 9–29. Wait-State Generation Conditions

	Wait State Generated	
Register	**Reads**	**Writes**
HPIA	No	HOM – No SAM – Yes
HPIC	No	1 to DSPINT/HINT – Yes All other cycles – No
HPID	HOM – No SAM – Yes	HOM – No SAM – Yes

Example Access Sequences

A complete host access cycle always involves two bytes, the first with HBIL low, and the second with HBIL high. This 2-byte sequence must be followed regardless of the type of host access (HPIA, HPIC, or data access) and the host must not break the first byte/second byte (HBIL low/high) sequence of an ongoing HPI access. If this sequence is broken, data may be lost, and unpredictable operation may result.

Before accessing data, the host must first initialize HPIC, in particular the BOB bit, and then HPIA (in this order, because BOB affects the HPIA access). After initializing BOB, the host can then write to HPIA with the correct byte alignment. On an HPI memory read operation, after completion of the HPIA write, the HPI memory is read and the contents at the given address are transferred to the two 8-bit data latches, the first byte data latch and the second byte data latch. Table 9–30 illustrates the sequence involved in initializing BOB and HPIA for an HPI memory read. In this example, BOB is set to 0 and a read is requested of the first HPI memory location (in this case 1000h), which contains FFFEh.

Table 9–30. Initialization of BOB and HPIA

Event	HD	HR/$\overline{W}$	HCNTL1/0	HBIL	HPIC	HPIA	latch1	latch2
Host writes HPIC, 1st byte	00	0	00	0	00xx	xxxx	xxxx	xxxx
Host writes HPIC, 2nd byte	00	0	00	1	0000	xxxx	xxxx	xxxx
Host writes HPIA, 1st byte	10	0	10	0	0000	10xx	xxxx	xxxx
Host writes HPIA, 2nd byte	00	0	10	1		1000	xxxx	xxxx
Internal HPI RAM read complete						1000	FF	FE

In the cycle shown in Table 9–30, BOB and HPIA are initialized, and by loading HPIA, an internal HPI memory access is initiated. The last line of Table 9–30 shows the condition of the HPI after the internal RAM read is complete; that is, after some delay following the end of the host write of the second byte to HPIA, the read is completed and the data has been placed in the upper and lower byte data latches. For the host to actually retrieve this data, it must perform an additional read of HPID. During this HPID read access, the contents of the first byte data latch appears on the HD pins when HBIL is low and the content of the second byte data latch appears on the HD pins when HBIL is high. Then the address is incremented if autoincrement is selected and the memory is read again into the data latches. Note that the address autoincrement occurs between the transfers of the first and second bytes. The sequence involved in this access is shown in Table 9–31.

Table 9–31. Read Access to HPI with Autoincrement

Event	HD	HR/$\overline{W}$	HCNTL1/0	HBIL	HPIC	HPIA	latch1	latch2
Host reads data, 1st byte	FF	1	01	0	0000	1000	FF	FE
Host reads data, 2nd byte	FE	1	01	1	0000	1001	FF	FE
Internal HPI RAM read complete						1001	6A	BC

In the access shown in Table 9–31, the data obtained from reading HPID is the data from the read initiated in the previous cycle (the one shown in Table 9–30) and the access performed as shown in Table 9–31 also initiates a further read, this time at location 1001h (because autoincrement was specified in this access by setting HCNTL1/0 to 01). Also, when autoincrement is selected, the increment occurs with each 16-bit word transferred (not with each byte); therefore, as shown in Table 9–31, the HPIA is incremented by only 1. The last line of Table 9–31 indicates that after the second internal RAM read is complete, the contents of location 1001h (6ABCh) has been read and placed into the upper and lower byte data latches.

During a write access to the HPI, the first byte data latch is overwritten by the data coming from the host while the HBIL pin is low, and the second byte data latch is overwritten by the data coming from the host while the HBIL pin is high. At the end of this write access, the data in both data latches is transferred as a 16-bit word to the HPI memory at the address specified by the HPIA register. The address is incremented prior to the memory write because autoincrement is selected.

An HPI write access is illustrated in Table 9–32. In this example, after the internal portion of the write is completed, location 1002h of HPI RAM contains 1234h. If a read of the same address follows this write, the same data just written in the data latches (1234h) is read back.

Table 9–32. Write Access to HPI with Auto-Increment

Event	HD	HR/$\overline{W}$	HCNTL1/0	HBIL	HPIC	HPIA	latch1	latch2
Host writes data, 1st byte	12	0	01	0	0000	1001	12	FE
Host writes data, 2nd byte	34	0	01	1	0000	1002	12	34
Internal HPI RAM write complete						1002	12	34

9.10.4 DSPINT and HINT Function Operation

The host and the ’C5x can interrupt each other using bits in the HPIC register. This subsection presents more information about this process.

Host Device Using DSPINT to Interrupt the ’C5x

A ’C5x interrupt is generated when the host writes a 1 to the DSPINT bit in HPIC. This interrupt can be used to wake up the ’C5x from IDLE. The host and the ’C5x always read this bit as 0. A ’C5x write has no effect. Once a 1 is written to DSPINT by the host, a 0 need not be written before another interrupt can be generated, and writing a 0 to this bit has no effect. The host should not write a 1 to the DSPINT bit while writing to BOB or HINT, or an unwanted ’C5x interrupt is generated.

On the ’C5x, the host-to-’C5x interrupt vector address is 18h. This interrupt is located in bit 11 of the IMR/IFR. Since the ’C5x interrupt vectors can be remapped into the HPI memory, the host can instruct the ’C5x to execute preprogrammed functions by simply writing the start address of a function to address 19h in the HPI memory prior to interrupting the ’C5x with a branch instruction located at address 18h.

Host Port Interface ('C5x) Using HINT to Interrupt the Host Device

When the 'C5x writes a 1 to the HINT bit in HPIC, the $\overline{\text{HINT}}$ output is driven low; the HINT bit is then read as a 1 by the 'C5x or the host. The $\overline{\text{HINT}}$ signal can be used to interrupt the host device. The host device, after detecting the $\overline{\text{HINT}}$ interrupt line, can acknowledge and clear the 'C5x interrupt and the HINT bit by writing a 1 to the HINT bit. The HINT bit is cleared and then read as a 0 by the 'C5x or the host, and the $\overline{\text{HINT}}$ pin is driven high. If the 'C5x or the host writes a 0, the HINT bit remains unchanged. While accessing the SMOD bit, the 'C5x should not write a 1 to the HINT bit unless it also wants to interrupt the host.

9.10.5 Considerations in Changing HPI Memory Access Mode (SAM/HOM) and IDLE2 Use

The HPI host-only mode (HOM) allows the host to access HPI RAM while the 'C5x is in IDLE2 (that is, completely halted). Additionally, the external clock input to the 'C5x can be stopped for the lowest power consumption configuration. Under these conditions, random accesses can still be made without having to restart the external clock for each access and wait for its lockup time if the 'C5x on-chip PLL is used. The external clock need only be restarted before taking the 'C5x out of IDLE2.

The host cannot access HPI RAM in SAM when the 'C5x is in IDLE2, because CPU clocks are required for access in this mode of operation. Therefore, if the host requires access to the HPI RAM while the 'C5x is in IDLE2, the 'C5x must change HPI mode to HOM before entering IDLE2. When the HPI is in HOM, the 'C5x can access HPIC to toggle the SMOD bit or send an interrupt to the host, but cannot access the HPI RAM block; a 'C5x access to the HPI RAM is disregarded in HOM. In order for the 'C5x to again access the HPI RAM block, HPI mode must be changed to SAM after exiting IDLE2.

To select HOM, a 0 must be written to the SMOD bit in HPIC. To select SAM, a 1 must be written to SMOD. When changing between HOM and SAM, two considerations must be met for proper operation. First, the instruction immediately following the one that changes from SAM to HOM must not be an IDLE2. This is because in this case, due to the 'C5x pipeline and delays in the SAM to HOM mode switch, the IDLE2 takes effect before the mode switch occurs, causing the HPI to remain in SAM; therefore, no host accesses can occur.

The second consideration is that when changing from HOM to SAM, the instruction immediately following the one that changes from HOM to SAM cannot read the HPI RAM block. This requirement is due to the fact that the mode has not yet changed when the HPI RAM read occurs and the RAM read is

ignored because the mode switch has not yet occurred. HPI RAM writes are not included in this restriction because these operations occur much later in the pipeline, so it is possible to write to HPI RAM in the instruction following the one which changes from HOM to SAM.

On the host side, there are no specific considerations associated with the mode changes. For example, it is possible to have a third device wake up the ’C5x from IDLE2 and the ’C5x changing to SAM upon wake-up without a software handshake with the host. The host can continue accessing while the HPI mode changes. However, if the host accesses the HPI RAM while the mode is being changed, the actual mode change will be delayed until the host access is completed. In this case, a ’C5x access to the HPI memory is also delayed.

Table 9–33 illustrates the sequence of events involved in entering and exiting an IDLE2 state on the ’C5x when using the HPI. Throughout the process, the HPI is accessible to the host.

Table 9–33. Sequence of Entering and Exiting IDLE2

Host or Other Device	’C5x	Mode	’C5x clock
	Switches mode to HOM	HOM	Running
	Executes a NOP	HOM	Running
	Executes IDLE2 instruction	HOM	Running
May stop DSP clock	In IDLE2	HOM	Stopped or running
Turns on DSP clock if it was stopped†	In IDLE2	HOM	Running
Sends an interrupt to DSP	In IDLE2	HOM	Running
	’C5x wakes up from IDLE2	HOM	Running
	’C5x switches mode to SAM	SAM	Running

† Sufficient wake-up time must be ensured when the ’C5x on-chip PLL is used.

9.10.6 Access of HPI Memory During Reset

The ’C5x is not operational during reset, but the host can access the HPI, allowing program or data downloads to the HPI memory. When this capability is used, it is often convenient for the host to control the ’C5x reset input. The sequence of events for resetting the ’C5x and downloading a program to HPI memory while the ’C5x is in reset is summarized in Table 9–34 and corresponds to the reset of the ’C5x.

Initially, the host stops accessing the HPI at least six ’C5x periods before driving the ’C5x reset line low. The host then drives the ’C5x reset line low and can start accessing the HPI after a minimum of four ’C5x periods. The HPI mode is automatically set to HOM during reset, allowing high-speed program download. The ’C5x clock can even be stopped at this time; however, the clock must be running when the reset line falls and rises for proper reset operation of the ’C5x.

Once the host has finished downloading into HPI memory, the host stops accessing the HPI and drives the ’C5x reset line high. At least 20 ’C5x periods after the reset line rising edge, the host can again begin accessing the HPI. This number of periods corresponds to the internal reset delay of the ’C5x. The HPI mode is automatically set to SAM upon exiting reset.

On the ’C5x, the RAM and OVLY bits must be set to 1 after reset for the HPI memory to be mapped into ’C5x program and data space, as with other single-access RAM blocks. The host, however, can access the HPI memory regardless of the status of these two bits. Also, if the host writes a 1 to DSPINT while the ’C5x is in reset, the interrupt is lost when the ’C5x comes out of reset. The ’C5x warm boot can use the HPI memory and start execution from the lowest HPI address.

Table 9–34. HPI Operation During RESET

Host	’C5x	Mode	’C5x clock
Waits 6 ’C5x clock periods	Running	X	Running
Brings RESET low and waits 4 clocks	Goes into reset	HOM	Running
Can stop ’C5x clock	In reset	HOM	Stopped or Running
Writes program and/or data in HPI memory	In reset	HOM	Stopped or Running
Turns on DSP clock if it was stopped†	In reset	HOM	Running
Brings RESET high	In reset	HOM	Running
Waits 20 ’C5x clock periods	Comes out of reset	SAM	Running
Can access HPI	Running	SAM	Running

† Sufficient wake-up time must be ensured when the ’C5x on-chip PLL is used.

Appendix A

Pinouts and Signal Descriptions

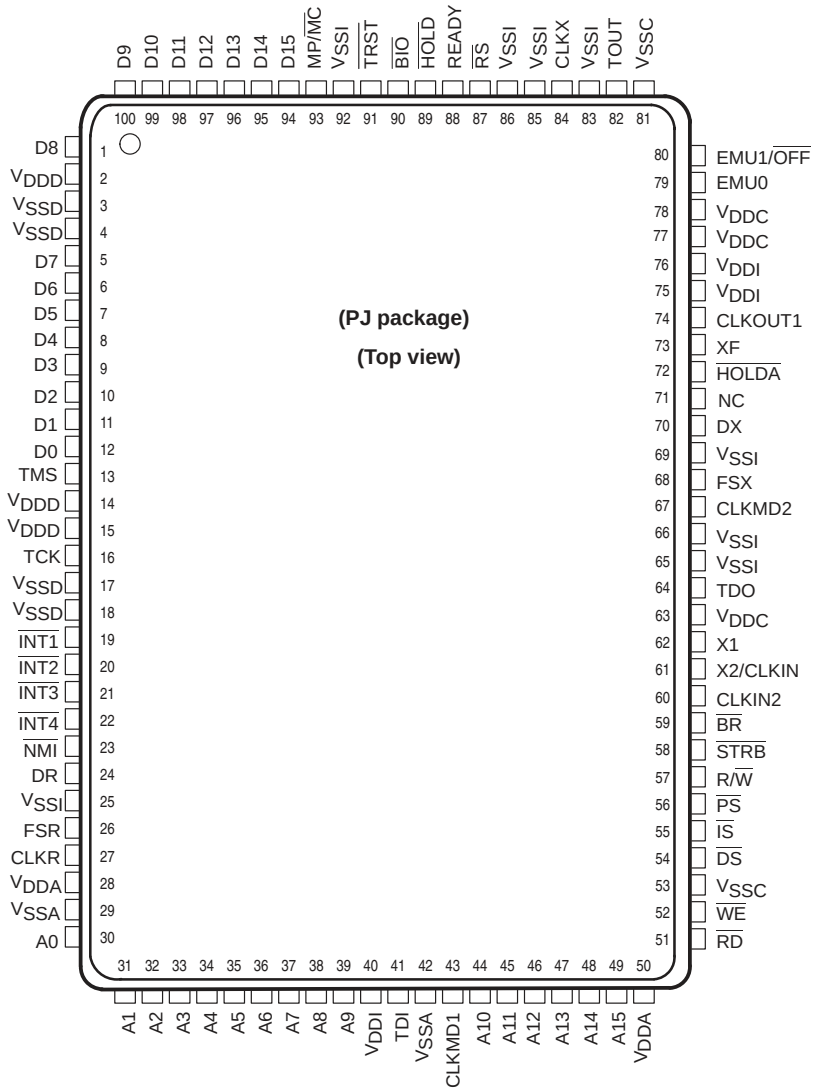
The TMS320C5x DSPs are available in a 100-pin quad flat-pack (QFP), 100-pin thin quad flat-pack (TQFP), 128-pin TQFP, 132-pin bumpered quad flat-pack (BQFP), and 144-pin TQFP packages. All packages conform to JEDEC specifications for electrical/electronic components. Refer to the figures and tables in this appendix for the pin/signal assignments of the different packages. Also, this appendix presents a table of signal definitions.

Topic **Page**

A.1 100-Pin QFP Pinout (’C52)

Refer to Figure A–1 and Table A–1 for pin/signal assignments of the ’C52 in the 100-pin QFP package.

Figure A–1. Pin/Signal Assignments for the ’C52 in 100-Pin QFP



Note: NC These pins are not connected (reserved).

Table A–1. Signal/Pin Assignments for the 'C52 in 100-Pin QFP

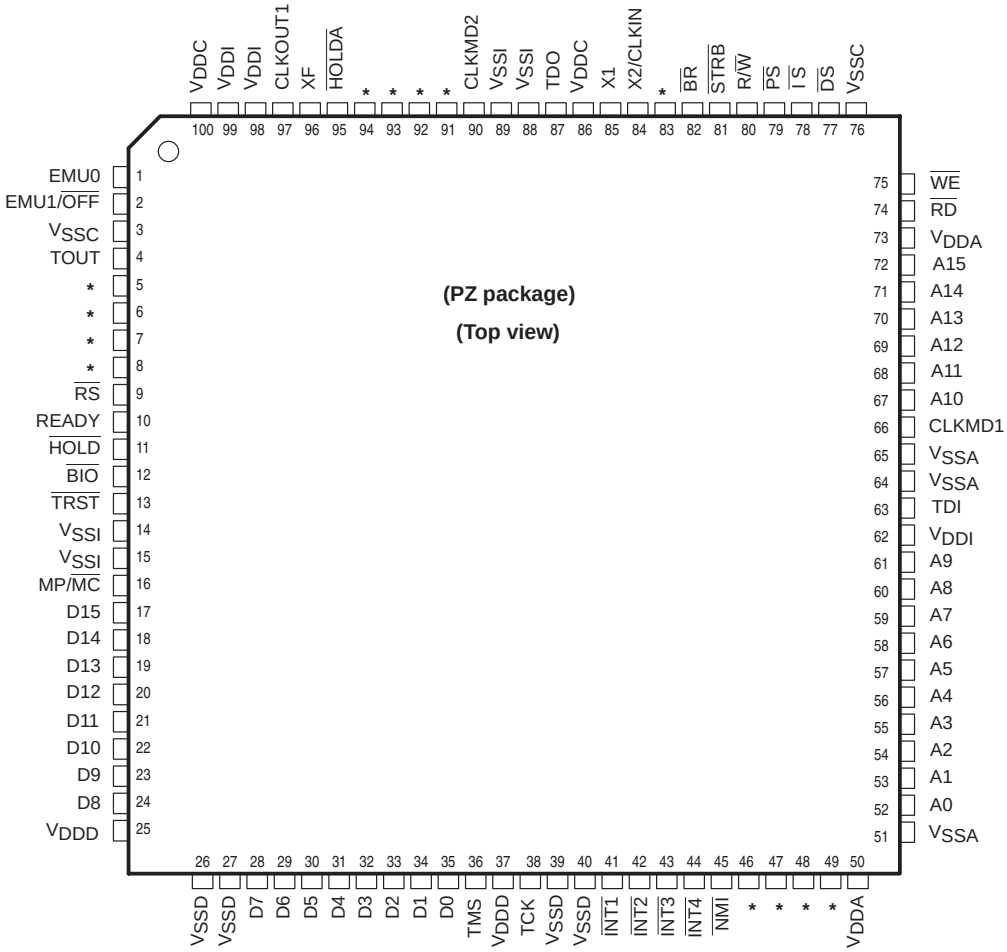
Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	30	D6	6	R/$\overline{W}$	57	V_{SSI}	69
A1	31	D7	5	$\overline{STRB}$	58	V_{SSI}	66
A2	32	D8	1	TCK	16	V_{SSI}	65
A3	33	D9	100	TDI	41	V_{SSI}	92
A4	34	D10	99	TDO	64	V_{SSI}	25
A5	35	D11	98	TMS	13	$\overline{WE}$	52
A6	36	D12	97	TOUT	82	X1	62
A7	37	D13	96	$\overline{TRST}$	91	X2/CLKIN1	61
A8	38	D14	95	V_{DDA}	28	XF	73
A9	39	D15	94	V_{DDA}	50	†	71
A10	44	DR	24	V_{DDC}	78		
A11	45	$\overline{DS}$	54	V_{DDC}	77		
A12	46	DX	70	V_{DDC}	63		
A13	47	EMU0	79	V_{DDD}	2		
A14	48	EMU1/$\overline{OFF}$	80	V_{DDD}	14		
A15	49	FSR	26	V_{DDD}	15		
$\overline{BR}$	59	FSX	68	V_{DDI}	40		
$\overline{BIO}$	90	$\overline{HOLD}$	89	V_{DDI}	75		
CLKIN2	60	$\overline{HOLDA}$	72	V_{DDI}	76		
CLKMD1	43	$\overline{INT1}$	19	V_{SSA}	42		
CLKMD2	67	$\overline{INT2}$	20	V_{SSA}	29		
CLKOUT1	74	$\overline{INT3}$	21	V_{SSC}	53		
CLKR	27	$\overline{INT4}$	22	V_{SSC}	81		
CLKX	84	$\overline{IS}$	55	V_{SSD}	3		
D0	12	MP/$\overline{MC}$	93	V_{SSD}	4		
D1	11	$\overline{NMI}$	23	V_{SSD}	17		
D2	10	$\overline{PS}$	56	V_{SSD}	18		
D3	9	READY	88	V_{SSI}	83		
D4	8	$\overline{RD}$	51	V_{SSI}	85		
D5	7	$\overline{RS}$	87	V_{SSI}	86		

† This pin is not connected (reserved).

A.2 100-Pin TQFP Pinout (’C51, ’C52, ’C53S, and ’LC56)

Refer to Figure A–2 and Table A–2 for pin/signal assignments of the ’C51, ’C52, ’C53S, and ’LC56 in the 100-pin TQFP package. Table A–6 on page A-12 lists device-specific pin/signal assignments for the ’C51, ’C52, ’C53S, and ’LC56.

Figure A–2. Pin/Signal Assignments for the ’C51, ’C52, ’C53S, and ’LC56 in 100-Pin TQFP



Note: * These pins are reserved for specific devices (see Table A–6 on page A-12).

Table A–2. Signal/Pin Assignments for the 'C51, 'C52, 'C53S, and 'LC56 in 100-Pin TQFP

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	52	D9	23	V_{DDA}	50	*	8
A1	53	D10	22	V_{DDA}	73	*	46
A2	54	D11	21	V_{DDC}	86	*	47
A3	55	D12	20	V_{DDC}	100	*	48
A4	56	D13	19	V_{DDD}	25	*	49
A5	57	D14	18	V_{DDD}	37	*	83
A6	58	D15	17	V_{DDI}	62	*	91
A7	59	DS	77	V_{DDI}	98	*	92
A8	60	EMU0	1	V_{DDI}	99	*	93
A9	61	EMU1/$\overline{OFF}$	2	TOUT	4	*	94
A10	67	$\overline{HOLD}$	11	V_{SSA}	65		
A11	68	$\overline{HOLDA}$	95	V_{SSA}	64		
A12	69	$\overline{INT1}$	41	V_{SSA}	51		
A13	70	$\overline{INT2}$	42	V_{SSC}	3		
A14	71	$\overline{INT3}$	43	V_{SSC}	76		
A15	72	$\overline{INT4}$	44	V_{SSD}	26		
$\overline{BR}$	82	$\overline{IS}$	78	V_{SSD}	27		
$\overline{BIO}$	12	MP/$\overline{MC}$	16	V_{SSD}	39		
CLKMD1	66	$\overline{NMI}$	45	V_{SSD}	40		
CLKMD2	90	$\overline{PS}$	79	V_{SSI}	14		
CLKOUT1	97	READY	10	V_{SSI}	15		
D0	35	$\overline{RD}$	74	V_{SSI}	88		
D1	34	$\overline{RS}$	9	V_{SSI}	89		
D2	33	R/$\overline{W}$	80	$\overline{WE}$	75		
D3	32	$\overline{STRB}$	81	X1	85		
D4	31	TCK	38	X2/CLKIN	84		
D5	30	TDI	63	XF	96		
D6	29	TDO	87	*	5		
D7	28	TMS	36	*	6		
D8	24	$\overline{TRST}$	13	*	7		

Legend: * These pins are reserved for specific devices (see Table A–6 on page A-12).

A.3 128-Pin TQFP Pinout (’LC57)

Refer to Figure A–3 and Table A–3 for pin/signal assignments of the ’LC57 in the 128-pin TQFP package.

Figure A–3. Pin/Signal Assignments for the ’LC57 in 128-Pin TQFP

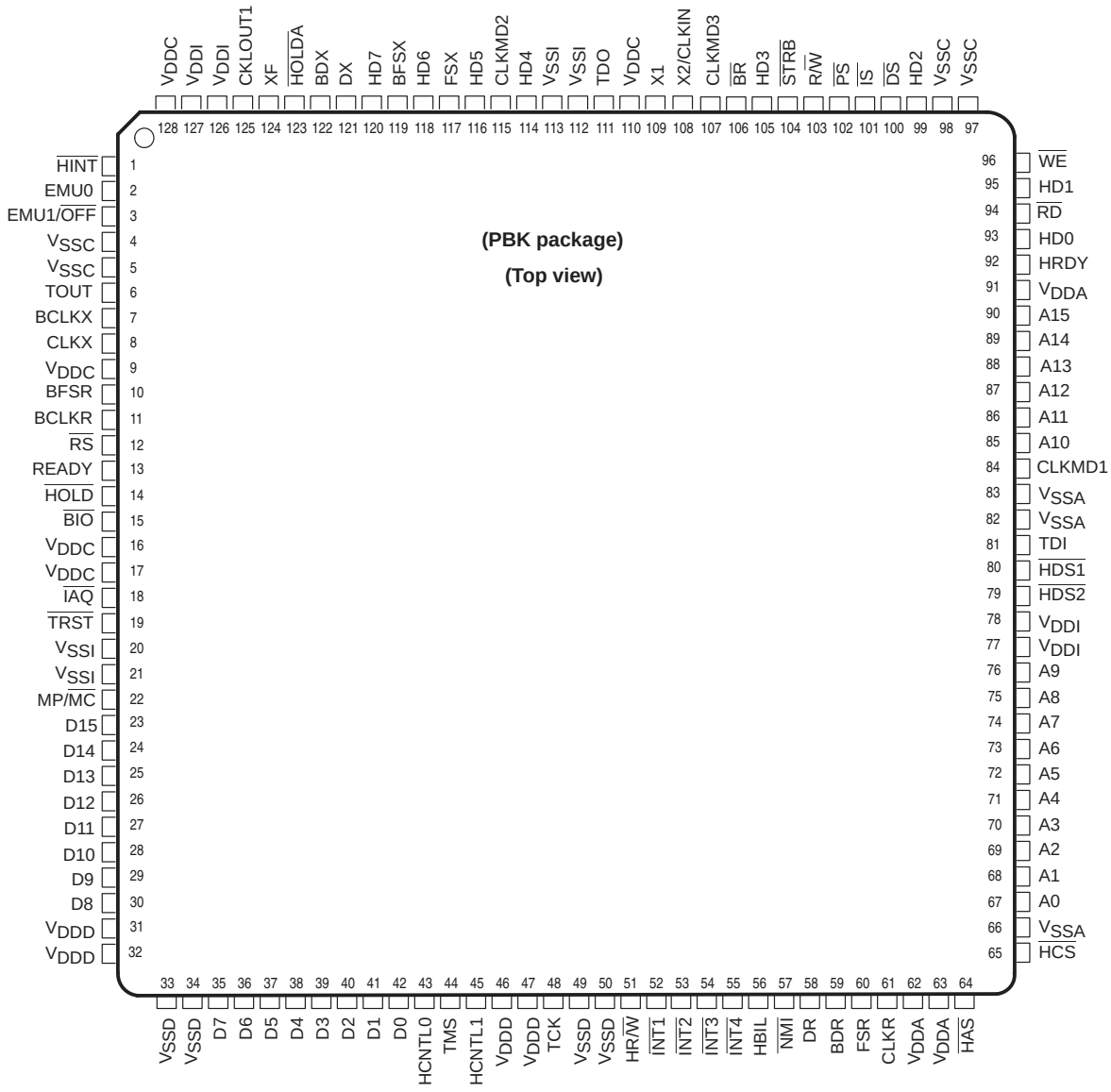


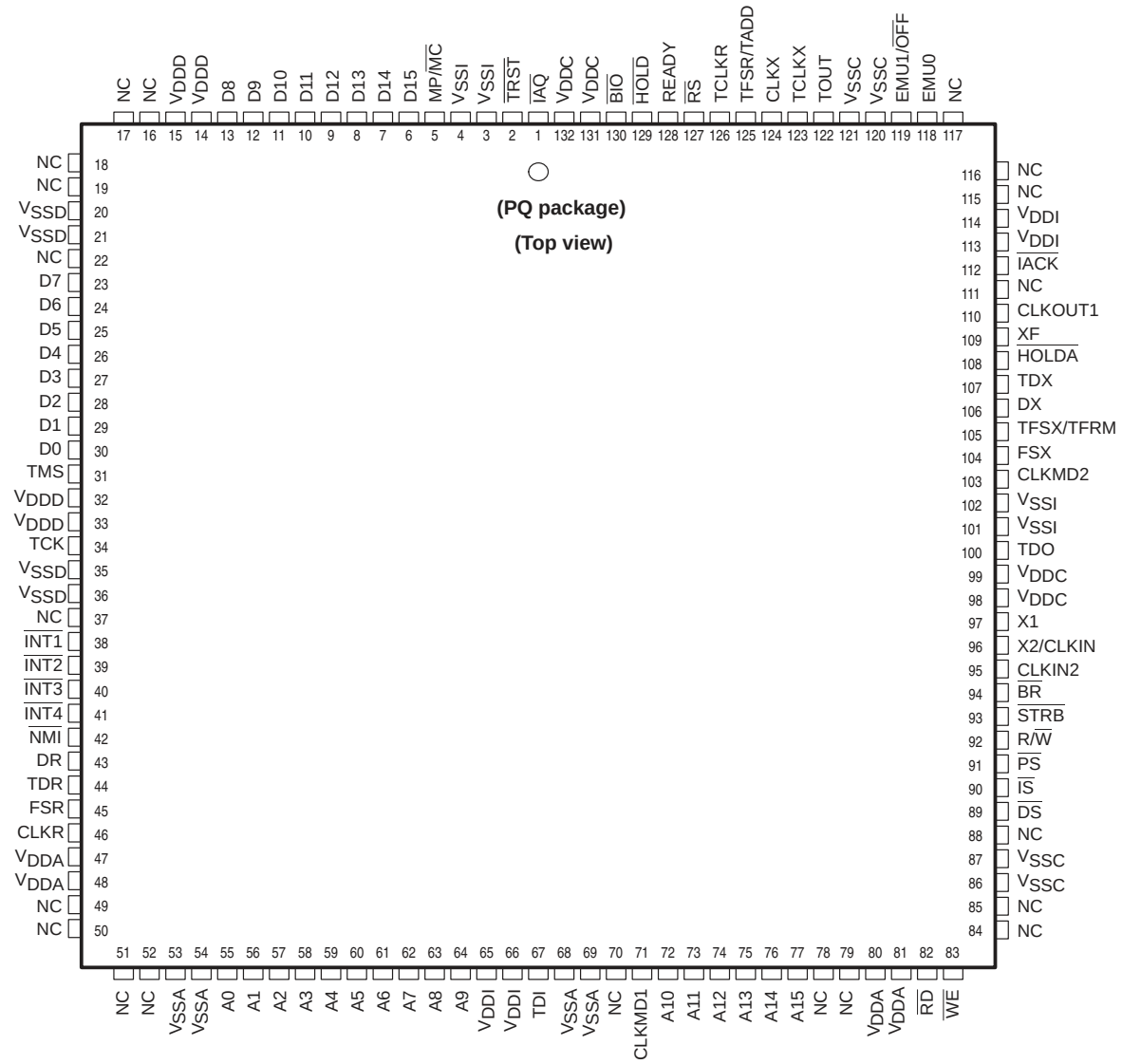
Table A–3. Signal/Pin Assignments for the 'LC57 in 128-Pin TQFP

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	67	CLKX	8	HD1	95	TDO	111	V_{SSD}	50
A1	68	D0	42	HD2	99	TMS	44	V_{SSI}	20
A2	69	D1	41	HD3	105	TOUT	6	V_{SSI}	21
A3	70	D2	40	HD4	114	$\overline{TRST}$	19	V_{SSI}	112
A4	71	D3	39	HD5	116	V_{DDC}	9	V_{SSI}	113
A5	72	D4	38	HD6	118	V_{DDA}	91	$\overline{WE}$	96
A6	73	D5	37	HD7	120	V_{DDA}	63	X1	109
A7	74	D6	36	$\overline{HDS1}$	80	V_{DDA}	62	XF	124
A8	75	D7	35	$\overline{HDS2}$	79	V_{DDC}	16		
A9	76	D8	30	$\overline{HINT}$	1	V_{DDC}	17		
A10	85	D9	29	$\overline{HOLD}$	14	V_{DDC}	110		
A11	86	D10	28	$\overline{HOLDA}$	123	V_{DDC}	128		
A12	87	D11	27	HRDY	92	V_{DDD}	31		
A13	88	D12	26	HR/$\overline{W}$	51	V_{DDD}	32		
A14	89	D13	25	$\overline{IAQ}$	18	V_{DDD}	46		
A15	90	D14	24	$\overline{INT1}$	52	V_{DDD}	47		
BCLKR	11	D15	23	$\overline{INT2}$	53	V_{DDI}	77		
BCLKX	7	DR	58	$\overline{INT3}$	54	V_{DDI}	78		
BDR	59	$\overline{DS}$	100	$\overline{INT4}$	55	V_{DDI}	126		
BDX	122	DX	121	$\overline{IS}$	101	V_{DDI}	127		
BFSR	10	EMU0	2	MP/$\overline{MC}$	22	V_{SSA}	66		
BFSX	119	EMU1/$\overline{OFF}$	3	$\overline{NMI}$	57	V_{SSA}	82		
$\overline{BIO}$	15	FSR	60	$\overline{PS}$	102	V_{SSA}	83		
$\overline{BR}$	106	FSX	117	$\overline{RD}$	94	V_{SSC}	4		
X2/CLKIN	108	$\overline{HAS}$	64	READY	13	V_{SSC}	5		
CLKMD1	84	HBIL	56	$\overline{RS}$	12	V_{SSC}	97		
CLKMD2	115	HCNTL0	43	R/$\overline{W}$	103	V_{SSC}	98		
CLKMD3	107	HCNTL1	45	$\overline{STRB}$	104	V_{SSD}	33		
CLKOUT1	125	$\overline{HCS}$	65	TCK	48	V_{SSD}	34		
CLKR	61	HD0	93	TDI	81	V_{SSD}	49		

A.4 132-Pin BQFP Pinout (’C50, ’C51, and ’C53)

Refer to Figure A–4 and Table A–4 for pin/signal assignments of the ’C50, ’C51, and ’C53 in the 132-pin BQFP package.

Figure A–4. Pin/Signal Assignments for the ’C50, ’C51, and ’C53 in 132-Pin BQFP



Note: NC These pins are not connected (reserved).

Table A–4. Signal/Pin Assignments for the 'C50, 'C51, and 'C53 in 132-Pin BQFP

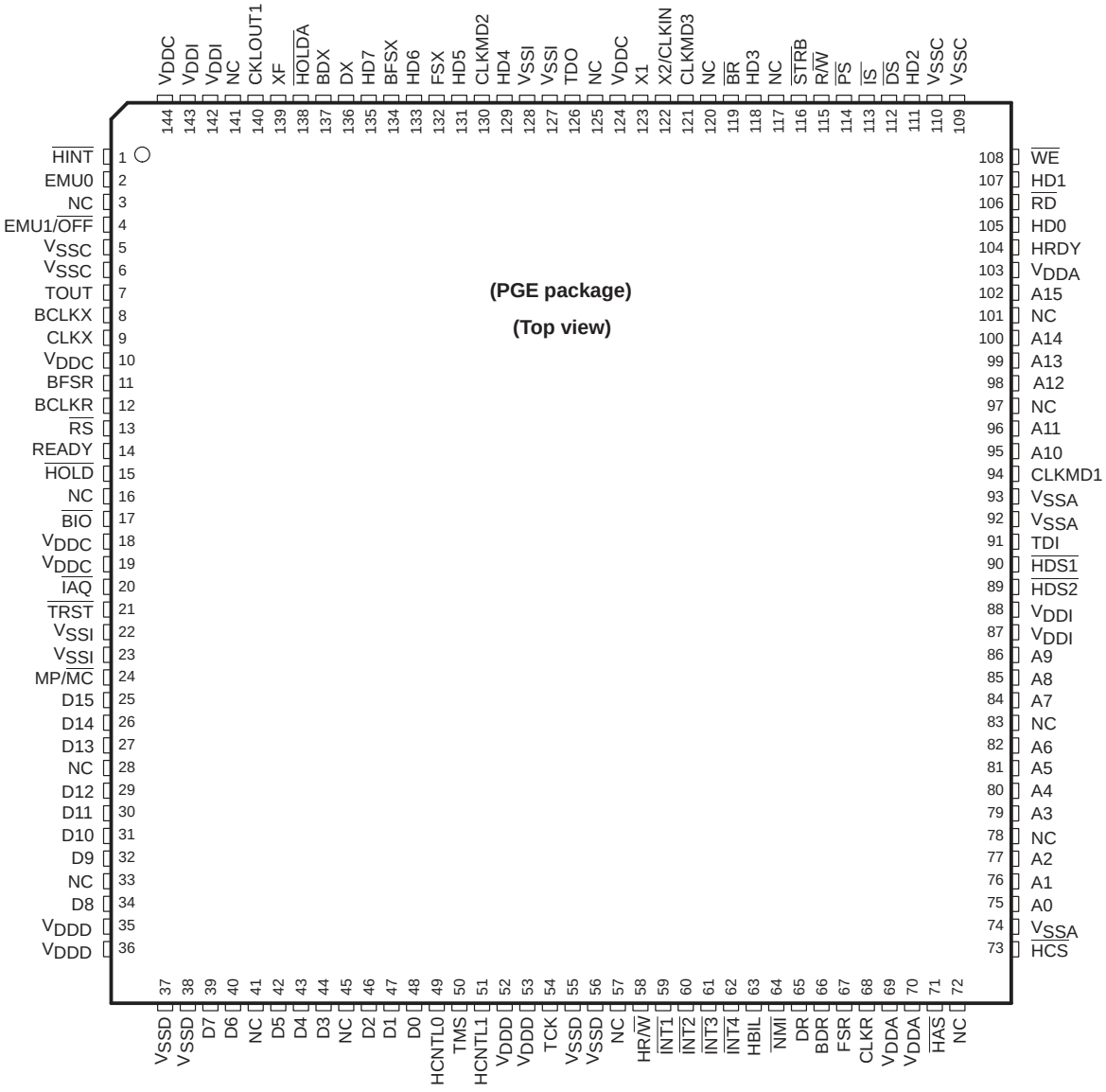
Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	55	D6	24	$\overline{RD}$	82	V_{DDI}	65	†	51
A1	56	D7	23	$\overline{RS}$	127	V_{DDI}	66	†	52
A2	57	D8	13	R/$\overline{W}$	92	V_{SSA}	53	†	70
A3	58	D9	12	$\overline{STRB}$	93	V_{SSA}	54	†	78
A4	59	D10	11	TCK	34	V_{SSA}	68	†	79
A5	60	D11	10	TCLKR	126	V_{SSA}	69	†	84
A6	61	D12	9	TCLKX	123	V_{SSC}	86	†	85
A7	62	D13	8	TDI	67	V_{SSC}	87	†	88
A8	63	D14	7	TDO	100	V_{SSC}	121	†	111
A9	64	D15	6	TDR	44	V_{SSC}	120	†	115
A10	72	DR	43	TDX	107	V_{SSD}	20	†	116
A11	73	$\overline{DS}$	89	TMS	31	V_{SSD}	21	†	117
A12	74	DX	106	TOUT	122	V_{SSD}	35		
A13	75	EMU0	118	$\overline{TRST}$	2	V_{SSD}	36		
A14	76	EMU1/$\overline{OFF}$	119	TFSR/TADD	125	V_{SSI}	3		
A15	77	FSR	45	TFSX/TFRM	105	V_{SSI}	4		
$\overline{BR}$	94	FSX	104	V_{DDC}	131	V_{SSI}	101		
$\overline{BIO}$	130	$\overline{HOLD}$	129	V_{DDC}	132	V_{SSI}	102		
CLKIN2	95	$\overline{HOLDA}$	108	V_{DDA}	47	$\overline{WE}$	83		
CLKMD1	71	$\overline{IACK}$	112	V_{DDA}	48	X1	97		
CLKMD2	103	$\overline{IAQ}$	1	V_{DDA}	80	X2/CLKIN	96		
CLKOUT1	110	$\overline{INT1}$	38	V_{DDA}	81	XF	109		
CLKR	46	$\overline{INT2}$	39	V_{DDC}	98	†	16		
CLKX	124	$\overline{INT3}$	40	V_{DDC}	99	†	17		
D0	30	$\overline{INT4}$	41	V_{DDD}	14	†	18		
D1	29	$\overline{IS}$	90	V_{DDD}	15	†	19		
D2	28	MP/$\overline{MC}$	5	V_{DDD}	32	†	22		
D3	27	$\overline{NMI}$	42	V_{DDD}	33	†	37		
D4	26	$\overline{PS}$	91	V_{DDI}	113	†	49		
D5	25	READY	128	V_{DDI}	114	†	50		

† These pins are not connected (reserved).

A.5 144-Pin TQFP Pinout (’C57S)

Refer to Figure A–5 and Table A–5 for pin/signal assignments of the ’C57S in the 144-pin TQFP package.

Figure A–5. Pin/Signal Assignments for the ’C57S in 144-Pin TQFP



Note: NC These pins are not connected (reserved).

Table A–5. Signal/Pin Assignments for the 'C57S in 144-Pin TQFP

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	75	CLKX	9	HD0	105	TCK	54	V_{SSD}	37
A1	76	D0	48	HD1	107	TDI	91	V_{SSD}	38
A2	77	D1	47	HD2	111	TDO	126	V_{SSD}	55
A3	79	D2	46	HD3	118	TMS	50	V_{SSD}	56
A4	80	D3	44	HD4	129	TOUT	7	V_{SSI}	22
A5	81	D4	43	HD5	131	$\overline{TRST}$	21	V_{SSI}	23
A6	82	D5	42	HD6	133	V_{DDA}	69	V_{SSI}	127
A7	84	D6	40	HD7	135	V_{DDA}	70	V_{SSI}	128
A8	85	D7	39	$\overline{HDS1}$	90	V_{DDA}	103	$\overline{WE}$	108
A9	86	D8	34	$\overline{HDS2}$	89	V_{DDC}	10	X1	123
A10	95	D9	32	$\overline{HINT}$	1	V_{DDC}	18	X2/CLKIN	122
A11	96	D10	31	$\overline{HOLD}$	15	V_{DDC}	19	XF	139
A12	98	D11	30	$\overline{HOLDA}$	138	V_{DDC}	124	†	3
A13	99	D12	29	HRDY	104	V_{DDC}	144	†	16
A14	100	D13	27	HR/$\overline{W}$	58	V_{DDD}	35	†	28
A15	102	D14	26	$\overline{IAQ}$	20	V_{DDD}	36	†	33
BCLKR	12	D15	25	$\overline{INT1}$	59	V_{DDD}	52	†	41
BCLKX	8	DR	65	$\overline{INT2}$	60	V_{DDD}	53	†	45
BDR	66	$\overline{DS}$	112	$\overline{INT3}$	61	V_{DDI}	87	†	57
BDX	137	DX	136	$\overline{INT4}$	62	V_{DDI}	88	†	72
BFSR	11	EMU0	2	$\overline{IS}$	113	V_{DDI}	142	†	78
BFSX	134	EMU1/$\overline{OFF}$	4	MP/$\overline{MC}$	24	V_{DDI}	143	†	83
$\overline{BIO}$	17	FSR	67	$\overline{NMI}$	64	V_{SSA}	74	†	97
$\overline{BR}$	119	FSX	132	$\overline{PS}$	114	V_{SSA}	92	†	101
CLKMD1	94	$\overline{HAS}$	71	$\overline{RD}$	106	V_{SSA}	93	†	117
CLKMD2	130	HBIL	63	READY	14	V_{SSC}	5	†	120
CLKMD3	121	HCNTL0	49	$\overline{RS}$	13	V_{SSC}	6	†	125
CLKOUT1	140	HCNTL1	51	R/$\overline{W}$	115	V_{SSC}	109	†	141
CLKR	68	$\overline{HCS}$	73	$\overline{STRB}$	116	V_{SSC}	110		

† These pins are not connected (reserved).

A.6 100-Pin TQFP Device-Specific Pinouts

Table A–6 lists device-specific pin/signal assignments for the ’C51, ’C52, ’C53S, and ’LC56 in the 100-pin TQFP package. Refer to Figure A–2 on page A-4 and Table A–2 on page A-5 for common pin/signal assignments of the ’C51, ’C52, ’C53S, and ’LC56.

Table A–6. Device-Specific Pin/Signal Assignments for the ’C51, ’C52, ’C53S, and ’LC56 in 100-Pin TQFP

Pin	’C51	’C52	’C53S	’LC56†
5	TCLKX	V_{SSI}	CLKX2	BCLKX
6‡	CLKX	CLKX	CLKX1	CLKX
7	TFSR/TADD	V_{SSI}	FSR2	BFSR
8	TCLKR	V_{SSI}	CLKR2	BCLKR
46‡	DR	DR	DR1	DR
47	TDR	V_{SSI}	DR2	BDR
48‡	FSR	FSR	FSR1	FSR
49‡	CLKR	CLKR	CLKR1	CLKR
83	CLKIN2	CLKIN2	CLKIN2	CLKMD3
91‡	FSX	FSX	FSX1	FSX
92	TFSX/TFRM	V_{SSI}	FSX2	BFSX
93‡	DX	DX	DX1	DX
94	TDX	NC§	DX2	BDX

† Pin names beginning with B indicate signals on the buffered serial port (BSP).
‡ No change in function.
§ NC = These pins are not connected (reserved).

A.7 Signal Descriptions

Table A–7 through Table A–18 list each signal, specifies the signal's operating state(s), and describes the signal's function.

Table A–7. Address and Data Bus Signal Descriptions

Signal	State	Description
A15 (MSB) A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 (LSB)	I/O/Z	**Parallel, bidirectional, 3-state address bus A15 (MSB) through A0 (LSB).** Multiplexed to address external data/program memory or I/O. Placed in high-impedance state in hold mode or when $\overline{\text{OFF}}$ † is active (low). These signals are used as inputs for external DMA access of the on-chip single-access RAM. They become inputs while $\overline{\text{HOLDA}}$ is active low, if the $\overline{\text{BR}}$ pin is externally driven low.
D15 (MSB) D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 (LSB)	I/O/Z	**Parallel, bidirectional, 3-state data bus D15 (MSB) through D0 (LSB).** Multiplexed to transfer data between the core CPU and external data/program memory or I/O devices. Placed in high-impedance state when not outputting, when $\overline{\text{RS}}$ or $\overline{\text{HOLD}}$ is asserted, or when $\overline{\text{OFF}}$ † is active (low). These signals are also used in external DMA access of the on-chip single-access RAM.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor. The $\overline{\text{BR}}$ pin has an internal pullup for performing DMA to the on-chip RAM.

Legend:
- I Input
- O Output
- Z High impedance
- † The $\overline{\text{OFF}}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–8. Memory Control Signal Descriptions

Signal	State	Description
$\overline{DS}$ $\overline{PS}$ $\overline{IS}$	O/Z	**Data, program, and I/O space select signals.** Always high unless low level is asserted for communicating to a particular external space. Placed into a high-impedance state in hold mode or when $\overline{OFF}$ † is active (low).
$\overline{RD}$	O/Z	**Read select.** Indicates an active, external read cycle and may connect directly to the output enable ($\overline{OE}$) of external devices. This signal is active on all external program, data, and I/O reads. It is placed into high-impedance state in hold mode or when $\overline{OFF}$ † is active (low).
READY	I	**Data ready input.** Indicates whether an external device is prepared for the bus transaction to be completed. If the device is not ready (READY is low), the processor waits one cycle and checks READY again. READY also indicates a bus grant to an external device after a $\overline{BR}$ (bus request) signal.
R/$\overline{W}$	I/O/Z	**Read/write signal.** Indicates transfer direction during communication to an external device. Normally in read mode (high), unless low level is asserted for performing a write operation. Placed in high-impedance state in hold mode or when $\overline{OFF}$ † is active (low). R/$\overline{W}$ is also used in external DMA access of the on-chip RAM cell. While $\overline{HOLDA}$ and $\overline{IAQ}$ are active (low), this signal indicates the direction of the data bus for DMA reads (high) and writes (low).
$\overline{STRB}$	I/O/Z	**Strobe signal.** Always high unless asserted low to indicate an external bus cycle. Placed in high-impedance state in the hold mode or when $\overline{OFF}$ † is active (low). $\overline{STRB}$ is also used in external DMA access of the on-chip single-access RAM. While $\overline{HOLDA}$ and $\overline{IAQ}$ are active (low), this signal is used to select the memory access.
$\overline{WE}$	O/Z	**Write enable.** The falling edge of this signal indicates that the device is driving the external data bus (D15–D0). Data can be latched by an external device on the rising edge of $\overline{WE}$. This signal is active on all external program, data, and I/O writes. It is placed into high-impedance state in hold mode or when $\overline{OFF}$ † is active (low).

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor. The $\overline{BR}$ pin has an internal pullup for performing DMA to the on-chip RAM.

Legend:
- I Input
- O Output
- Z High impedance
- † The $\overline{OFF}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–9. Multiprocessing Signal Descriptions

Signal	State	Description
$\overline{\text{BIO}}$	I	**Branch control input.** Can be used to control a conditional branch instruction. $\overline{\text{BIO}}$ is sampled during the fetch of the conditional branch instruction.
$\overline{\text{BR}}$	I/O/Z	**Bus request signal.** Asserted during access of external global data memory space. READY is asserted to the device when the global data memory is available for the bus transaction. $\overline{\text{BR}}$ can be used to extend the data memory address space by up to 32K words. It goes into high impedance when $\overline{\text{OFF}}$ † is active (low). $\overline{\text{BR}}$ is used in external DMA access of the on-chip single-access RAM. While $\overline{\text{HOLDA}}$ is active low, $\overline{\text{BR}}$ is externally driven low to request access to the on-chip single-access RAM.
$\overline{\text{HOLD}}$	I	**Hold input.** This signal is asserted to request control of the address, data, and control lines. When $\overline{\text{HOLDA}}$ is acknowledged by the 'C5x, these lines go to the high-impedance state.
$\overline{\text{HOLDA}}$	O/Z	**Hold acknowledge signal.** Indicates to the external circuitry that the processor is in a hold state. At the same time, the address, data, and memory control lines are in a high-impedance state so that they are available to the external circuitry for access of local memory. This signal also goes into high impedance when $\overline{\text{OFF}}$ † is active (low).
$\overline{\text{IACK}}$	O/Z	**Interrupt acknowledge signal.** Indicates receipt of an interrupt and that the program counter is fetching the interrupt vector location designated by A15–A0. This signal goes into high impedance when $\overline{\text{OFF}}$ † is active (low).
$\overline{\text{IAQ}}$	O/Z	**Instruction acquisition signal.** This signal is asserted (active) when there is an instruction address on the address bus. This signal goes into high impedance when $\overline{\text{OFF}}$ † is active (low). $\overline{\text{IAQ}}$ is also used in external DMA access of the on-chip single-access RAM. While $\overline{\text{HOLDA}}$ is active low, $\overline{\text{IAQ}}$ acknowledges the $\overline{\text{BR}}$ request for access of the on-chip single-access RAM and stops indicating instruction acquisition.
XF	O/Z	**External flag output** (latched software-programmable signal). This signal is set high or low by specific instructions or by loading status register 1 (ST1). XF can be used for signaling other processors in multiprocessor configurations or as a general-purpose output pin. This signal also goes into high impedance when $\overline{\text{OFF}}$ † is active (low). This pin is set high at reset.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor. The $\overline{\text{BR}}$ pin has an internal pullup for performing DMA to the on-chip RAM.

Legend:
- I Input
- O Output
- Z High impedance
- † The $\overline{\text{OFF}}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–10. Initialization, Interrupt, and Reset Operations Signal Descriptions

Signal	State	Description
$\overline{INT4}$ $\overline{INT3}$ $\overline{INT2}$ $\overline{INT1}$	I	**External user interrupt inputs.** Prioritized and maskable by the interrupt mask register (IMR) and interrupt mode (INTM) bit in status register 0 (ST0). Can be polled and reset via the interrupt flag register (IFR).
MP/$\overline{MC}$	I	**Microprocessor/microcomputer mode select pin.** If active (low) at reset (microcomputer mode), the pin causes the internal program ROM to be mapped into program memory space. In the microprocessor mode, all program memory is mapped externally. This pin is sampled only during reset, and the mode is set at reset. The mode can be overridden via the software control bit MP/$\overline{MC}$ in the PMST register.
$\overline{NMI}$	I	**Nonmaskable interrupt.** External interrupt that cannot be masked via the IMR or the INTM bit in ST0. When $\overline{NMI}$ is activated, the processor traps to the appropriate vector location.
$\overline{RS}$	I	**Reset input.** Causes the device to terminate execution and forces the program counter to 0. When $\overline{RS}$ is brought to a high level, execution begins at location 0h of program memory. $\overline{RS}$ affects various registers and status bits.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.
Legend: I Input

Table A–11. Supply Signal Descriptions

Signal	State	Description
V_{DDA}	S	Power supply for address bus.
V_{DDC}	S	Power supply for memory control signals.
V_{DDD}	S	Power supply for data bus.
V_{DDI}	S	Power supply for inputs and internal logic.
V_{SSA}	S	Ground for address bus.
V_{SSC}	S	Ground for memory control signals.
V_{SSD}	S	Ground for data bus.
V_{SSI}	S	Ground for inputs and internal logic.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.
Legend: S Supply

Table A–12. Oscillator/Timer Signal Descriptions

Signal	State	Description
CLKIN2†	I	**External clock input.** Divide-by-1 input clock for driving the internal machine rate. PLL clock input for ’C50, ’C51, ’C52, ’C53, and ’C53S.
CLKOUT1	O/Z	**Master clock output signal.** This signal cycles at the machine-cycle rate of the CPU. The internal machine cycle is bounded by the rising edges of this signal. This signal goes into high impedance when $\overline{\text{OFF}}$ ‡ is active (low).
TOUT	O	**Timer output.** This pin issues a pulse when the on-chip timer counts down past zero. The pulse is a CLKOUT1 cycle wide.
X1	O	**Internal oscillator output.** Output pin from the internal oscillator for the crystal. If the internal oscillator is not used, this pin should be left unconnected. This signal does not go into high impedance when $\overline{\text{OFF}}$ ‡ is active (low).
X2/CLKIN	I	**External clock input.** Input pin to internal oscillator from the crystal. If the internal oscillator is not being used, a clock may be input to the device on this pin. PLL clock input for ’LC56, ’C57S, and ’LC57.
CLKMD1	I	**Clock mode input.** Input pin to determine clock mode. Refer to Table A–13 and Table A–14 for the clock options.
CLKMD2	I	**Clock mode input.** Input pin to determine clock mode. Refer to Table A–13 and Table A–14 for the clock options.
CLKMD3§	I	**Clock mode input.** Input pin to determine clock mode. Refer to Table A–14 for the clock options.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.

Legend:
- I Input
- O Output
- Z High impedance
- † This pin is not available on the ’LC56, ’C57S, and ’LC57 devices.
- ‡ The $\overline{\text{OFF}}$ signal, when active (low), puts all ’C5x output drivers into the high-impedance state.
- § This pin is not available on the ’C50, ’C51, ’C52, ’C53, and ’C53S devices.

Table A–13. Oscillator/Timer Standard Options ('C50, 'C51, C52, 'C53, and 'C53S Only)

Signal	State	CLKMD1	CLKMD2	Clock Mode
CLKMD1 CLKMD2	I	0	0	❑ PLL disabled ❑ Internal oscillator disabled ❑ Input clock provided to X2/CLKIN pin ❑ External clock with divide-by-2 option
		0	1	❑ Reserved for test purposes
		1	0	❑ PLL enabled ❑ Internal oscillator disabled ❑ Input clock provided to CLKIN2 pin ❑ External clock option: *For 'C50, 'C51, 'C53, and 'C53S: multiply-by-1 option* *For 'C52: multiply-by-2 option*
		1	1	❑ PLL disabled ❑ Internal oscillator enabled ❑ Input clock provided to X2/CLKIN pin ❑ Internal or external divide-by-2 option

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.
Legend: I Input

Table A–14. Oscillator/Timer Expanded Options ('LC56, 'C57S, and 'LC57 Only)

Signal	State	CLKMD1	CLKMD2	CLKMD3	Clock Mode
CLKMD1 CLKMD2 CLKMD3	I	0	0	0	❑ PLL enabled ❑ Internal oscillator disabled ❑ External multiply-by-3 option
		0	0	1	❑ Internal oscillator disabled ❑ External divide-by-2 option
		0	1	0	❑ PLL enabled ❑ Internal oscillator disabled ❑ External multiply-by-4 option
		0	1	1	❑ PLL enabled ❑ Internal oscillator disabled ❑ External multiply-by-2 option
		1	0	0	❑ PLL enabled ❑ Internal oscillator disabled ❑ External multiply-by-5 option
		1	0	1	❑ PLL enabled ❑ Internal oscillator disabled ❑ External multiply-by-1 option
		1	1	0	❑ PLL enabled ❑ Internal oscillator disabled ❑ External multiply-by-9 option
		1	1	1	❑ Internal oscillator enabled ❑ External/internal divide-by-2 option

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.
Legend: I Input

Table A–15. Serial Port Interface Signal Descriptions

Signal	State	Description
CLKR CLKR1 CLKR2 TCLKR	I	**Receive clock signal.** External clock signal for clocking data into the data receive shift register (RSR) from the data receive pin(s): ❑ 'C50, 'C51, or 'C53: pins DR or TDR. ❑ 'C53S: pins DR1 or DR2. ❑ 'C51, 'C52, 'LC56, 'C57S, or 'LC57: pin DR. The receive clock signal must be present during serial port transfers. If the serial port is not being used, the pin(s) can be sampled as an input via the IN0 bit in the serial port control register (SPC and/or TSPC).
CLKX CLKX1 CLKX2 TCLKX	I/O/Z	**Transmit clock signal.** Clock signal for clocking data from the data transmit shift register (XSR) to the data transmit pin(s): ❑ 'C50, 'C51, or 'C53: pins DX or TDX. ❑ 'C53S: pins DX1 or DX2. ❑ 'C51, 'C52, 'LC56, 'C57S, or 'LC57: pin DX. The clock signal can be an input if the MCM bit in the SPC (TSPC) is cleared. This pin may also be driven by the device at 1/4 the CLKOUT1 frequency when the MCM bit is set. If the serial port is not being used, the pin(s) can be sampled as an input via the IN1 bit in the SPC (and/or TSPC). The signal(s) go into high impedance when the $\overline{\text{OFF}}$ † signal is active (low).
DR DR1 DR2 TDR	I	**Received serial data.** The RSR receives serial data through this input pin.
DX DX1 DX2 TDX	O/Z	**Transmitted serial data.** The XSR transmits serial data through this pin. This pin is placed in a high-impedance state when not transmitting and also when the $\overline{\text{OFF}}$ † signal is active (low).
FSR FSR1 FSR2 TFSR/ TADD	I/O/Z	**Receive frame synchronization.** The falling edge of these pulses initiates the data receive process, beginning the clocking of the RSR. TFSR becomes an input/output (TADD) pin when the 'C50, 'C51, and the 'C53 are operating in TDM mode (TDM bit=1). In TDM mode, this pin is used to output/input the address of the port. This signal goes into high impedance when $\overline{\text{OFF}}$ † is active (low).
FSX FSX1 FSX2 TFSX/ TFRM	I/O/Z	**Transmit frame synchronization.** The falling edge of these pulses initiates the data transmit process, beginning the clocking of the XSR. Following reset, the default operating condition of these pulses is an input. This pin can be selected by software to be an output when the TXM bit in the SPC (TSPC) is set. This signal goes into high impedance when the $\overline{\text{OFF}}$ † signal is active (low). When the 'C50, 'C51, and 'C53 are operating in the TDM mode (TDM bit=1), the TFSX pin becomes TFRM (TDM frame synch).

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.

Legend:
- I Input
- O Output
- Z High impedance
- † The $\overline{\text{OFF}}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–16. Buffered Serial Port Interface Signal Descriptions ('LC56 and 'C57 Only)

Signal	State	Description
BCLKR	I	**Receive clock signal.** External clock signal for clocking data into the data receive shift register (BRSR) from the data receive (BDR) pin. Data is clocked on the falling edge of BCLKR, if the CLKP bit in the BSP control extension register (SPCE) is cleared; data is clocked on the rising edge of BCLKR, if the CLKP bit is set. If the serial port is not used, this pin can be sampled as an input via the IN0 bit in the SPC.
BCLKX	I/O/Z	**Transmit clock signal.** Clock signal for clocking data from the data transmit shift register (BXSR) to the data transmit (BDX) pin. Data is clocked on the rising edge of CLKX, if the CLKP bit in the SPCE is cleared; data is clocked on the falling edge of CLKX, if the CLKP bit is set. CLKX can be an input if the MCM bit in the BSP control register (BSPC) is cleared. When the MCM bit of BSPC is set to 1, CLKX is driven by an on-chip source having a frequency equal to 1/(CLKDV+1) of CLKOUT. CLKDV value is defined in SPCE. When CLKDV is odd or equal to 0, the CLKX duty cycle is 50%. When CLKDV is an even value (CLKDV=2p), the CLKX high and low state durations depend on CLKP. When CLKP is 0, the high state duration is p+1 cycles and the low state duration is p cycles; when CLKP is 1, the high state duration is p cycles and the low state duration is p+1 cycles. Following device reset, the default operating condition of CLKX is an input. If the serial port is not used, this pin can be sampled as an input via the IN1 bit in the SPC.
BDR	I	**Received serial data.** The BRSR receives serial data through this input pin.
BDX	O/Z	**Transmitted serial data.** The BXSR transmits serial data through this pin. This pin is placed in a high-impedance state when not transmitting and also when the $\overline{\text{OFF}}$ † signal is active (low).
BFSR	O	**Receive frame synchronization.** This signal initiates the data receive process. Upon reset, BFSR is active high. BFSR can be configured as active low by setting the FSP bit in the SPCE.
BFSX	I/O/Z	**Transmit frame synchronization.** This signal initiates the data transmit process. Upon reset, BFSX is an input signal. BFSX can be configured as an output by setting the TXM bit in the SPC. Upon reset, BFSX is active high. BFSX can be configured as active low by setting the FSP bit in the SPCE.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.

Legend:
I Input
O Output
Z High impedance
† The $\overline{\text{OFF}}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–17. Host Port Interface Signal Descriptions ('C57 Only)

Signal	State	Description
$\overline{HAS}$	I	**Address strobe input.** Hosts with a multiplexed address and data bus connect $\overline{HAS}$ to their ALE pin or equivalent. HBIL, HCNTL0/1, and HR/$\overline{W}$ are then latched on $\overline{HAS}$ falling edge. When used, $\overline{HAS}$ must precede the later of $\overline{HCS}$, $\overline{HDS1}$, or $\overline{HDS2}$ (see 'C5x data sheet for detailed HPI timing specifications). Hosts with separate address and data bus can connect $\overline{HAS}$ to a logic-1 level. In this case, HBIL, HCNTL0/1, and HR/$\overline{W}$ are latched by the later of $\overline{HDS1}$, $\overline{HDS2}$, or $\overline{HCS}$ falling edge while $\overline{HAS}$ stays inactive (high).
HBIL	I	**Byte identification input.** Identifies first or second byte of transfer (but not most significant or least significant — this is specified by the BOB bit in the HPIC register, described later in this section). HBIL is low for the first byte and high for the second byte.
HCNTL0 HCNTL1	I	**Control inputs.** Indicates type of host access to the HPI address register (HPIA), the HPI data latches (with optional address increment), or the HPI control register (HPIC). **HCNTL1 / HCNTL0 / Description** 0 / 0 / Host can read or write the HPIC. 0 / 1 / Host can read or write the HPI data latches. ❑ HPIA postincremented when data is read ❑ HPIA preincremented when data is written 1 / 0 / Host can read or write the HPIA. ❑ HPIA points to the HPI memory. ❑ DSP does not have access to the HPIA. 1 / 1 / Host can read or write the HPI data latches**.** HPIA is not affected.
$\overline{HCS}$	I	**Chip select input.** Serves as the enable input for the HPI and must be low during an access but may stay low between accesses. $\overline{HCS}$ normally precedes $\overline{HDS1}$ and $\overline{HDS2}$, but this signal also samples HCNTL0/1, HR/$\overline{W}$, and HBIL if $\overline{HAS}$ is not used and $\overline{HDS1}$ or $\overline{HDS2}$ are already low.
HD7 (MSB) HD6 HD5 HD4 HD3 HD2 HD1 HD0 (LSB)	I/O/Z	**Parallel, bidirectional, 3-state data bus.** HD7 (MSB) through HD0 (LSB) are placed in high-impedance state when not outputting ($\overline{HDSx}$ and $\overline{HCS}$ = 1) or when $\overline{OFF}$ † is active (low).

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.

Legend: I Input; O Output; Z High impedance; S Supply

† The $\overline{OFF}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–17. Host Port Interface Signal Descriptions ('C57 Only) (Continued)

Signal	State	Description
$\overline{\text{HDS1}}$ $\overline{\text{HDS2}}$	I	**Data strobe input.** Control transfer of data during host access cycles. Also, when $\overline{\text{HAS}}$ is not used, used to sample HBIL, HCNTL0/1, and HR/$\overline{\text{W}}$ when $\overline{\text{HCS}}$ is already low (which is the case in normal operation). Hosts with separate read and write strobes connect those strobes to either $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$. Hosts with a single data strobe connect it to either $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$, connecting the unused pin high. Regardless of HDS connections, HR/$\overline{\text{W}}$ is still required to determine direction of transfer. Because $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ are internally exclusive-NORed, hosts with a high true data strobe can connect this to one of the HDS inputs with the other HDS input connected low.
$\overline{\text{HINT}}$	O/Z	**Host interrupt.** Controlled by the HINT bit in the HPIC. This pin driven high when the 'C5x is being reset. Placed in high impedance when $\overline{\text{OFF}}$ † is active (low).
HRDY	O/Z	**HPI ready output.** When high, indicates that the HPI is ready for a transfer to be performed. When low, indicates that the HPI is busy completing the internal portion of the previous transaction. Placed in high impedance when $\overline{\text{OFF}}$ † is active (low). $\overline{\text{HCS}}$ enables HRDY; that is, HRDY is always high when $\overline{\text{HCS}}$ is high.
HR/$\overline{\text{W}}$	I	**Read/write input.** Hosts must drive HR/$\overline{\text{W}}$ high to read HPI and low to write HPI. Hosts without a R/$\overline{\text{W}}$ strobe can use an address line.

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor.

Legend:
- I Input
- O Output
- Z High impedance
- S Supply
- † The $\overline{\text{OFF}}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Table A–18. Emulation/Testing Signal Descriptions

Signal	State	Description
TCK	I	**IEEE JTAG Standard 1149.1 test clock.** This is normally a free-running clock signal with a 50% duty cycle. The changes on test access port (TAP) input signals (TMS and TDI) are clocked into the TAP controller, instruction register, or selected test data register on the rising edge of TCK. Changes at the TAP output signal (TDO) occur on the falling edge of TCK.
TDI	I	**IEEE JTAG Standard 1149.1 test data input.** TDI is clocked into the selected register (instruction or data) on a rising edge of TCK.
TDO	O/Z	**IEEE JTAG Standard 1149.1 test data output.** The contents of the selected register (instruction or data) is shifted out of TDO on the falling edge of TCK. TDO is in the high-impedance state except when it is scanning data. This signal also goes into high impedance when $\overline{\text{OFF}}$ † is active low.
TMS	I	**IEEE JTAG Standard 1149.1 test mode select.** This serial control input is clocked into the TAP controller on the rising edge of TCK.
$\overline{\text{TRST}}$	I	**IEEE JTAG Standard 1149.1 test reset.** This signal, when active high, gives the IEEE Standard 1149.1 scan system control of the operations of the device. If this signal is not connected or driven low, the device operates in its functional mode, and the signals are ignored.
EMU0	I/O/Z	**Emulator pin 0.** When $\overline{\text{TRST}}$ is driven low, EMU0 must be high for activation of the $\overline{\text{OFF}}$ condition. When $\overline{\text{TRST}}$ is driven high, EMU0 is used as an interrupt to or from the emulator system. This pin is defined as input/output via IEEE Standard 1149.1 scan.
EMU1/$\overline{\text{OFF}}$	I/O/Z	**Emulator pin 1/disable all outputs.** When $\overline{\text{TRST}}$ is driven high, this pin is used as an interrupt to or from the emulator system and is defined as input/output via IEEE Standard 1149.1 scan. When $\overline{\text{TRST}}$ is driven low, this pin is configured as $\overline{\text{OFF}}$. The EMU1/$\overline{\text{OFF}}$ signal, when active (low), puts all output drivers into the high-impedance state. Note that $\overline{\text{OFF}}$ is used exclusively for testing and emulation purposes (not for multiprocessing applications). Thus, for the $\overline{\text{OFF}}$ condition, the following conditions must apply for at least 2 machine cycles: ❑ $\overline{\text{TRST}}$ = low ❑ EMU0 = high ❑ EMU1/$\overline{\text{OFF}}$ = low

Note: Input pins that are unused may be connected to V_{DD} or to an external pullup resistor. For emulation, $\overline{\text{TRST}}$ has an internal pulldown, and TMS, TCK, and TDI have internal pullups. EMU0 and EMU1 require external pullups to support emulation.

Legend:
I Input
O Output
Z High impedance
† The $\overline{\text{OFF}}$ signal, when active (low), puts all 'C5x output drivers into the high-impedance state.

Appendix B

Instruction Classes and Cycles

Instructions are classified into several categories, or classes, according to cycles required. This appendix describes the instruction classes. Because a single instruction can have multiple syntaxes and types of execution, it can appear in multiple classes.

The tables in this appendix show the number of cycles required for a given ’C5x instruction to execute in a given memory configuration when executed as a single instruction.

Topic **Page**

B.1 Cycle Class-to-Instruction Set Summary

Table B–1 provides a cycle class-to-instruction set cross reference. See Table 6–2 on page 6-4 for definitions of symbols and abbreviations used in the syntax expression.

Table B–1. Cycle Class-to-Instruction Set Summary

Cycle class	Cycle class description	Mnemonic†
Class I‡	1 word, 1 cycle, no memory operands	ABS, **ADCB**, ADD #k, **ADDB**, ADRK #k, **ANDB**, APAC, **BSAR**, **CLRC**, CMPL, CMPR, **CRGT**, **CRLT**, **EXAR**, IDLE, **IDLE2**, **LACB**, LACL #k, MAR, MPY #k, NEG, NOP, NORM, **ORB**, PAC, POP, PUSH, ROL, **ROLB**, ROR, **RORB**, **SACB**, **SATH**, **SATL**, **SBB**, **SBBB**, SBRK #k, **SETC**, SFL, **SFLB**, SFR, **SFRB**, SPAC, SPM, SUB #k, **XC**, **XORB**, ZAP, **ZPR**
Class IIA‡	1 word, 1 cycle, memory read operand	ADD, ADDC, ADDS, ADDT, AND, BIT, BITT, **CPL**, LACC, LACL, LACT, LPH, LT, LTA, LTP, LTS, MPY, MPYA, MPYS, MPYU, OR, PSHD, RPT, SQRA, SQRS, SUB, SUBB, SUBC, SUBS, SUBT, XOR, ZALR
Class IIB	1 word, 1 cycle, memory-mapped register read	**LAMM**
Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	ADD #lk, AND #lk, LACC #lk, LAR ARn, #lk, MPY #lk, OR #lk, RPT #lk, **RPTB**, **RPTZ** #lk, SUB #lk, XOR #lk
Class IVA	1 word, 1 cycle, memory write operand	POPD, SACH, SACL, SAR, SPH, SPL, SST
Class IVB	1 word, 1 cycle, memory-mapped register write	**SAMM**
Class V	1 word, 1 cycle, memory read and write	**APL**, DMOV, LTD, **OPL**, **XPL**
Class VI	2 words, 2 cycles, memory read and write	**APL**, **OPL**, **XPL**
Class VIIa	2 words, 2 cycles, memory read operand	**CPL** #lk
Class VIIb	2 words, 2 cycles, memory write operand, not repeatable	**SPLK**

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ ADD #k, ADRK #k, LACL #k, MPY #k, RPT, SBRK #k, SPM, SUB #k, and XC are not repeatable instructions.

Table B–1. Cycle Class-to-Instruction Set Summary (Continued)

Cycle class	Cycle class description	Mnemonic†
Class VIII	2 words, 4 cycles program counter discontinuity, no delayed slots, not repeatable	B, BANZ, **BCND**, CALL, **CC**
Class IX	2 words, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	BANZD, BCNDD, BD, CALLD, CCD
Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	BACC, CALA, **INTR**, NMI, RET, **RETC**, **RETE**, **RETI**, TRAP
Class XI	1 word, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	BACCD, CALAD, RETCD, RETD
Class XII	2 words, 3 cycles, block data transfer, data to data space	BLDD
Class XIII	1 word, 2 cycles, block data transfer, data to data space	BLDD
Class XIV	2 words, 3 cycles, block data transfer, program to data space	BLPD
Class XV	1 word, 2 cycles, block data transfer, program to data space	BLPD
Class XVI	1 word, 2 cycles, block data transfer, data to program space	**BLDP**
Class XVII	1 word, 3 cycles, table read	TBLR
Class XVIII	1 word, 3 cycles, table write	TBLW
Class XIX	2 words, 3 cycles, multiply accumulate	MAC
Class XX	1 word, 2 cycles, multiply accumulate	**MADS**
Class XXI	2 words, 3 cycles, multiply accumulate with data move	MACD

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.
‡ ADD #k, ADRK #k, LACL #k, MPY #k, RPT, SBRK #k, SPM, SUB #k, and XC are not repeatable instructions.

Table B–1. Cycle Class-to-Instruction Set Summary (Continued)

Cycle class	Cycle class description	Mnemonic†
Class XXII	1 word, 2 cycles, multiply accumulate with data move	**MADD**
Class XXIII	2 words, 2 cycles, memory-mapped register load	**LMMR**
Class XXIV	2 words, 2 cycles, memory-mapped register store	**SMMR**
Class XXV	2 words, 3 cycles, output port	OUT
Class XXVI	2 words, 2 cycles, input port	IN
Class XXVII	1 word, 2 cycles, pipeline-protected, memory read	LAR, LDP, LST
Class XXVIII	1 word, 2 cycles, pipeline-protected, memory read, not repeatable	LAR ARn, #k; LDP #k
Class XXIIX	1 word, 2 cycles, no memory access, not repeatable, pipeline protected	RPT #k

† **Bold** typeface indicates instructions that are new for the 'C5x instruction set.
‡ ADD #k, ADRK #k, LACL #k, MPY #k, RPT, SBRK #k, SPM, SUB #k, and XC are not repeatable instructions.

B.2 Instruction Set-to-Cycle Class Summary

Table B–2 provides an instruction set-to-cycle class cross reference. See Table 6–2 on page 6-4 for definitions of symbols and abbreviations used in the syntax expression.

Table B–2. Instruction Set-to-Cycle Class Summary

Mnemonic†	Cycle class	Cycle class description	Page
ABS	Class I	1 word, 1 cycle, no memory operands	6-28
ADCB	Class I	1 word, 1 cycle, no memory operands	6-30
ADD	Class IIA	1 word, 1 cycle, memory read operand	6-31
ADD #k	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-31
ADD #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-31
ADDB	Class I	1 word, 1 cycle, no memory operands	6-35
ADDC	Class IIA	1 word, 1 cycle, memory read operand	6-36
ADDS	Class IIA	1 word, 1 cycle, memory read operand	6-38
ADDT	Class IIA	1 word, 1 cycle, memory read operand	6-40
ADRK #k	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-42
AND	Class IIA	1 word, 1 cycle, memory read operand	6-43
AND #lk	Class III	2 words, 2 cycles long-immediate operand, no memory access, not repeatable	6-43
ANDB	Class I	1 word, 1 cycle, no memory operands	6-46
APAC	Class I	1 word, 1 cycle, no memory operands	6-47
APL	Class V	1 word, 1 cycle, memory read and write	6-48
APL	Class VI	2 words, 2 cycles, memory read and write	6-48
B	Class VIII	2 words, 4 cycles program counter discontinuity, no delayed slots, not repeatable	6-51
BACC	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-52
BACCD	Class XI	1 word, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-53

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
BANZ	Class VIII	2 words, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-54
BANZD	Class IX	2 words, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-56
BCND	Class VIII	2 words, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-58
BCNDD	Class IX	2 words, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-60
BD	Class IX	2 words, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-62
BIT	Class IIA	1 word, 1 cycle, memory read operand	6-63
BITT	Class IIA	1 word, 1 cycle, memory read operand	6-65
BLDD	Class XIII	1 word, 2 cycles, block data transfer, data to data space	6-67
BLDD	Class XII	2 words, 3 cycles, block data transfer, data to data space	6-67
BLDP	Class XVI	1 word, 2 cycles, block data transfer, data to program space	6-73
BLPD	Class XV	1 word, 2 cycles, block data transfer, program to data space	6-76
BLPD	Class XIV	2 words, 3 cycles, block data transfer, program to data space	6-76
BSAR	Class I	1 word, 1 cycle, no memory operands	6-82
CALA	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-83
CALAD	Class XI	1 word, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-84
CALL	Class VIII	2 words, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-85
CALLD	Class IX	2 words, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-86
CC	Class VIII	2 words, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-88

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
CCD	Class IX	2 words, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-90
CLRC	Class I	1 word, 1 cycle, no memory operands	6-92
CMPL	Class I	1 word, 1 cycle, no memory operands	6-94
CMPR	Class I	1 word, 1 cycle, no memory operands	6-95
CPL	Class IIA	1 word, 1 cycle, memory read operand	6-97
CPL #lk	Class VIIa	2 words, 2 cycles, memory read operand	6-97
CRGT	Class I	1 word, 1 cycle, no memory operands	6-100
CRLT	Class I	1 word, 1 cycle, no memory operands	6-102
DMOV	Class V	1 word, 1 cycle, memory read and write	6-104
EXAR	Class I	1 word, 1 cycle, no memory operands	6-106
IDLE	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-107
IDLE2	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-108
IN	Class XXVI	2 words, 2 cycles, input port	6-109
INTR	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-111
LACB	Class I	1 word, 1 cycle, no memory operands	6-113
LACC	Class IIA	1 word, 1 cycle, memory read operand	6-114
LACC #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-114
LACL	Class IIA	1 word, 1 cycle, memory read operand	6-117
LACL #k	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-117
LACT	Class IIA	1 word, 1 cycle, memory read operand	6-120
LAMM	Class IIB	1 word, 1 cycle, memory-mapped register read	6-122
LAR	Class XXVII	1 word, 2 cycles, pipeline-protected, memory read	6-124
LAR ARn, #k	Class XXVIII	1 word, 2 cycles, pipeline-protected, memory read, not repeatable	6-124

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
LAR ARn, #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-124
LDP	Class XXVII	1 word, 2 cycles, pipeline-protected, memory read	6-127
LDP #k	Class XXVIII	1 word, 2 cycles, pipeline-protected, memory read, not repeatable	6-127
LMMR	Class XXIII	2 words, 2 cycles, memory-mapped register load	6-130
LPH	Class IIA	1 word, 1 cycle, memory read operand	6-133
LST	Class XXVII	1 word, 2 cycles, pipeline-protected, memory read	6-135
LT	Class IIA	1 word, 1 cycle, memory read operand	6-138
LTA	Class IIA	1 word, 1 cycle, memory read operand	6-140
LTD	Class V	1 word, 1 cycle, memory read and write	6-142
LTP	Class IIA	1 word, 1 cycle, memory read operand	6-145
LTS	Class IIA	1 word, 1 cycle, memory read operand	6-147
MAC	Class XIX	2 words, 3 cycles, multiply accumulate	6-149
MACD	Class XXI	2 words, 3 cycles, multiply accumulate with data move	6-153
MADD	Class XXII	1 word, 2 cycles, multiply accumulate with data move	6-158
MADS	Class XX	1 word, 2 cycles, multiply accumulate	6-162
MAR	Class I	1 word, 1 cycle, no memory operands	6-166
MPY	Class IIA	1 word, 1 cycle, memory read operand	6-168
MPY #k	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-168
MPY #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-168
MPYA	Class IIA	1 word, 1 cycle, memory read operand	6-171
MPYS	Class IIA	1 word, 1 cycle, memory read operand	6-173
MPYU	Class IIA	1 word, 1 cycle, memory read operand	6-175
NEG	Class I	1 word, 1 cycle, no memory operands	6-177

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
NMI	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-179
NOP	Class I	1 word, 1 cycle, no memory operands	6-180
NORM	Class I	1 word, 1 cycle, no memory operands	6-181
OPL	Class V	1 word, 1 cycle, memory read and write	6-184
OPL	Class VI	2 words, 2 cycles, memory read and write	6-184
OR	Class IIA	1 word, 1 cycle, memory read operand	6-187
OR #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-187
ORB	Class I	1 word, 1 cycle, no memory operands	6-190
OUT	Class XXV	2 words, 3 cycles, output port	6-191
PAC	Class I	1 word, 1 cycle, no memory operands	6-193
POP	Class I	1 word, 1 cycle, no memory operands	6-194
POPD	Class IVA	1 word, 1 cycle, memory write operand	6-196
PSHD	Class IIA	1 word, 1 cycle, memory read operand	6-198
PUSH	Class I	1 word, 1 cycle, no memory operands	6-200
RET	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-202
RETC	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-203
RETCD	Class XI	1 word, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-205
RETD	Class XI	1 word, 2 cycles, program counter discontinuity, 2 delayed slots, not repeatable	6-207
RETE	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-208
RETI	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-209
ROL	Class I	1 word, 1 cycle, no memory operands	6-210

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
ROLB	Class I	1 word, 1 cycle, no memory operands	6-211
ROR	Class I	1 word, 1 cycle, no memory operands	6-212
RORB	Class I	1 word, 1 cycle, no memory operands	6-213
RPT	Class IIA	1 word, 1 cycle, memory read operand, not repeatable	6-214
RPT #k	Class XXIX	1 word, 2 cycle, no memory operands, not repeatable	6-214
RPT #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-214
RPTB	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-217
RPTZ #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-219
SACB	Class I	1 word, 1 cycle, no memory operands	6-220
SACH	Class IVA	1 word, 1 cycle, memory write operand	6-221
SACL	Class IVA	1 word, 1 cycle, memory write operand	6-223
SAMM	Class IVB	1 word, 1 cycle, memory-mapped register write	6-225
SAR	Class IVA	1 word, 1 cycle, memory write operand	6-227
SATH	Class I	1 word, 1 cycle, no memory operands	6-229
SATL	Class I	1 word, 1 cycle, no memory operands	6-231
SBB	Class I	1 word, 1 cycle, no memory operands	6-232
SBBB	Class I	1 word, 1 cycle, no memory operands	6-233
SBRK #k	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-234
SETC	Class I	1 word, 1 cycle, no memory operands	6-235
SFL	Class I	1 word, 1 cycle, no memory operands	6-237
SFLB	Class I	1 word, 1 cycle, no memory operands	6-238
SFR	Class I	1 word, 1 cycle, no memory operands	6-239
SFRB	Class I	1 word, 1 cycle, no memory operands	6-241
SMMR	Class XXIV	2 words, 2 cycles, memory-mapped register store	6-243

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
SPAC	Class I	1 word, 1 cycle, no memory operands	6-246
SPH	Class IVA	1 word, 1 cycle, memory write operand	6-247
SPL	Class IVA	1 word, 1 cycle, memory write operand	6-249
SPLK	Class VIIb	2 words, 2 cycles, memory write operand, not repeatable	6-251
SPM	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-252
SQRA	Class IIA	1 word, 1 cycle, memory read operand	6-253
SQRS	Class IIA	1 word, 1 cycle, memory read operand	6-255
SST	Class IVA	1 word, 1 cycle, memory write operand	6-257
SUB	Class IIA	1 word, 1 cycle, memory read operand	6-259
SUB #k	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-259
SUB #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-259
SUBB	Class IIA	1 word, 1 cycle, memory read operand	6-263
SUBC	Class IIA	1 word, 1 cycle, memory read operand	6-265
SUBS	Class IIA	1 word, 1 cycle, memory read operand	6-267
SUBT	Class IIA	1 word, 1 cycle, memory read operand	6-269
TBLR	Class XVII	1 word, 3 cycles, table read	6-271
TBLW	Class XVIII	1 word, 3 cycles, table write	6-274
TRAP	Class X	1 word, 4 cycles, program counter discontinuity, no delayed slots, not repeatable	6-277
XC	Class I	1 word, 1 cycle, no memory operands, not repeatable	6-278
XOR	Class IIA	1 word, 1 cycle, memory read operand	6-280
XOR #lk	Class III	2 words, 2 cycles, long-immediate operand, no memory access, not repeatable	6-280
XORB	Class I	1 word, 1 cycle, no memory operands	6-283
XPL	Class V	1 word, 1 cycle, memory read and write	6-284
XPL	Class VI	2 words, 2 cycles, memory read and write	6-284

† **Bold** typeface indicates instructions that are new for the ’C5x instruction set.

Table B–2. Instruction Set-to-Cycle Class Summary (Continued)

Mnemonic†	Cycle class	Cycle class description	Page
ZALR	Class IIA	1 word, 1 cycle, memory read operand	6-287
ZAP	Class I	1 word, 1 cycle, no memory operands	6-289
ZPR	Class I	1 word, 1 cycle, no memory operands	6-290

† **Bold** typeface indicates instructions that are new for the 'C5x instruction set.

Appendix C

System Migration

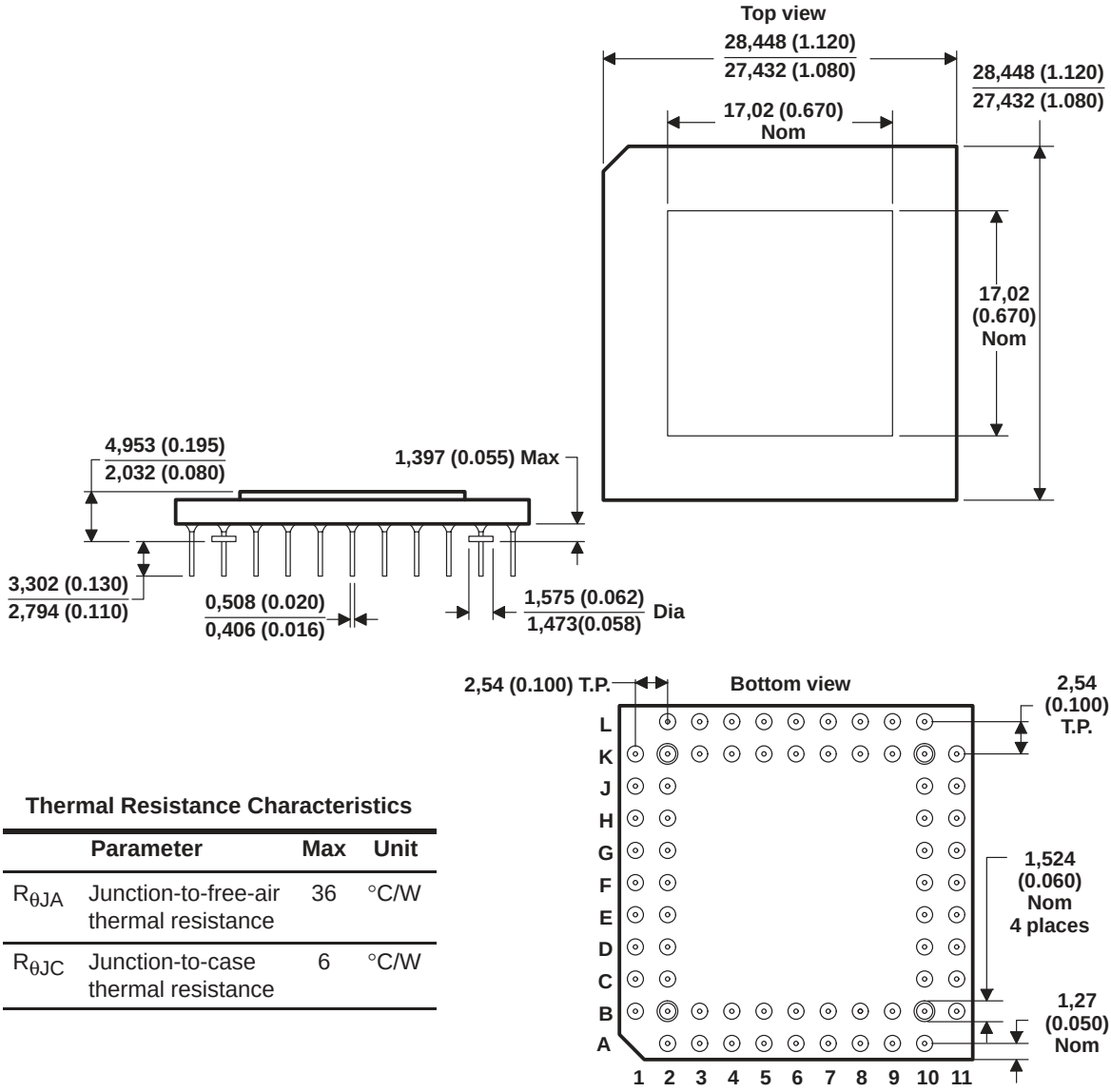
This appendix contains information that is necessary to upgrade a ’C2x system into a ’C5x system. The information consists of a detailed list of the programming differences and hardware and timing differences between the two generations of TMS320 DSPs. Note that the ’C50, C51, and ’C53 have the same features with the exception of memory map; so within this appendix, any reference to ’C5x applies to ’C50, ’C51, and ’C53, unless otherwise stated.

Topic **Page**

C.1 Package and Pin Layout

The ’C25 is available in both a 68-pin ceramic pin grid array (CPGA) as shown in Figure C–1 and a 68-pin plastic leaded chip carrier (PLCC) as shown in Figure C–2. The ’C5x devices are available in various packages as shown in Appendix A, *Pinouts and Signal Descriptions*.

Figure C–1. TMS320C25 in 68-Pin CPGA

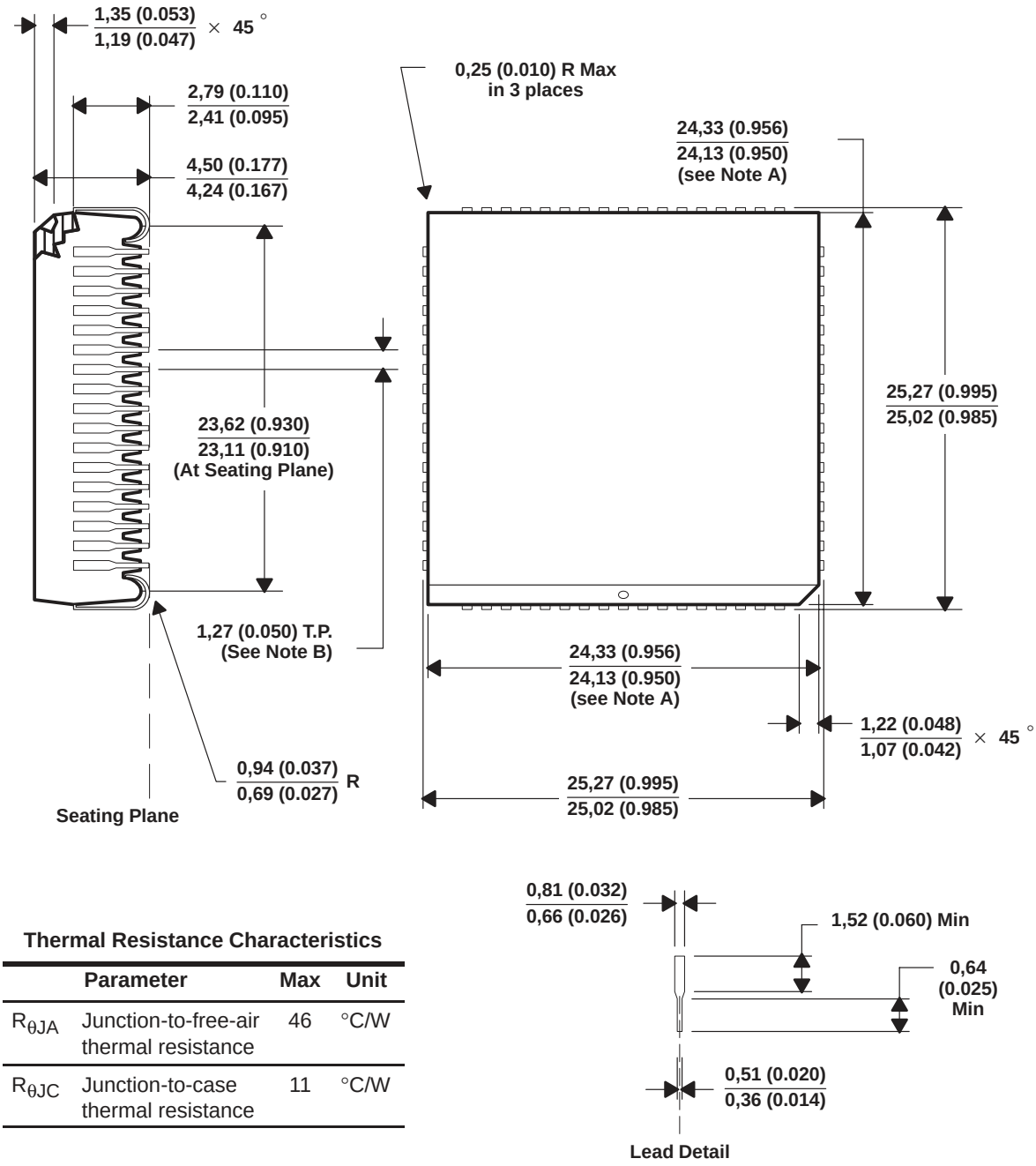


Thermal Resistance Characteristics

	Parameter	Max	Unit
$R_{\theta JA}$	Junction-to-free-air thermal resistance	36	°C/W
$R_{\theta JC}$	Junction-to-case thermal resistance	6	°C/W

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES.

Figure C–2. TMS320C25 in 68-Pin PLCC



Thermal Resistance Characteristics

	Parameter	Max	Unit
$R_{\theta JA}$	Junction-to-free-air thermal resistance	46	°C/W
$R_{\theta JC}$	Junction-to-case thermal resistance	11	°C/W

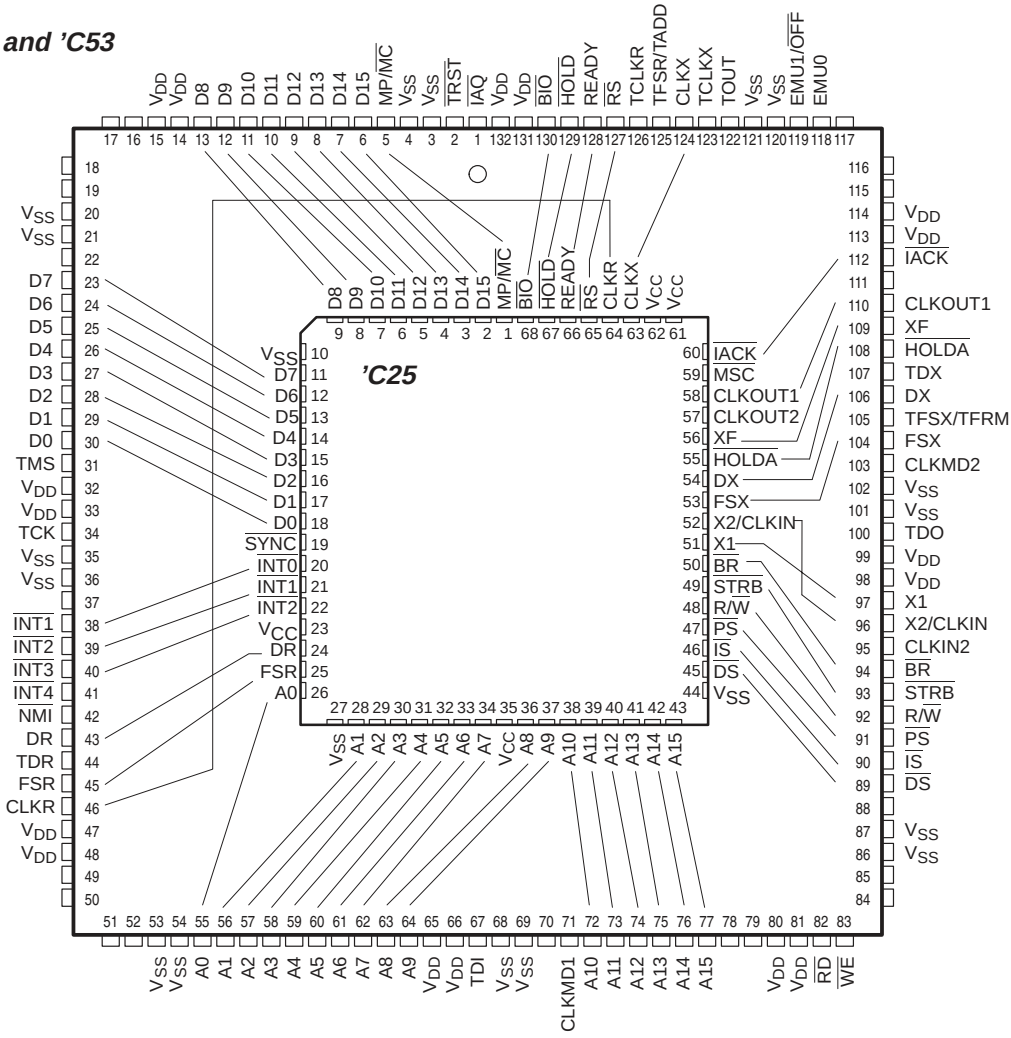
Notes: A. Centerline of center pin, each side, is within 0,10 (0.004) of package centerline as determined by this dimension.
B. Location of each pin is within 0,127 (0.005) of true position with respect to center pin on each side.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES.

When a ’C25 is upgraded to a ’C50, ’C51, or ’C53, there is minimal layout modification. The ’C5x signals are on the same side (except the CLKR and A0 pins), and in the same order (except the X1 and X2/CLKIN pins) as those of the ’C25. Figure C–3 shows the pin-to-pin relationship between the ’C25 and the ’C5x devices in J-leaded chip carrier packages. The two devices are not drawn to scale. The power (V_{DD}) and ground (V_{SS}) signals are symmetrically positioned on the ’C5x so that, in conjunction with the $\overline{OFF}$ signal, the device is not damaged by inserting it in the wrong orientation. The ’C5x has more power and ground pins to provide higher performance and more noise immunity than the ’C25.

Figure C–3. TMS320C25-to-TMS320C5x Pin/Signal Relationship

'C50, 'C51, and 'C53



Note: Pins without callouts are unassigned (reserved).

Three 'C25 signals (CLKOUT2, $\overline{\text{MSC}}$ and $\overline{\text{SYNC}}$) are not present on the 'C5x. Because the 'C5x operates with a divide-by-two clock, it can be synchronized with reset. Therefore, there is no need for the $\overline{\text{SYNC}}$ signal. With only two phases, there are no external timings that tie to the CLKOUT2 of the 'C25.

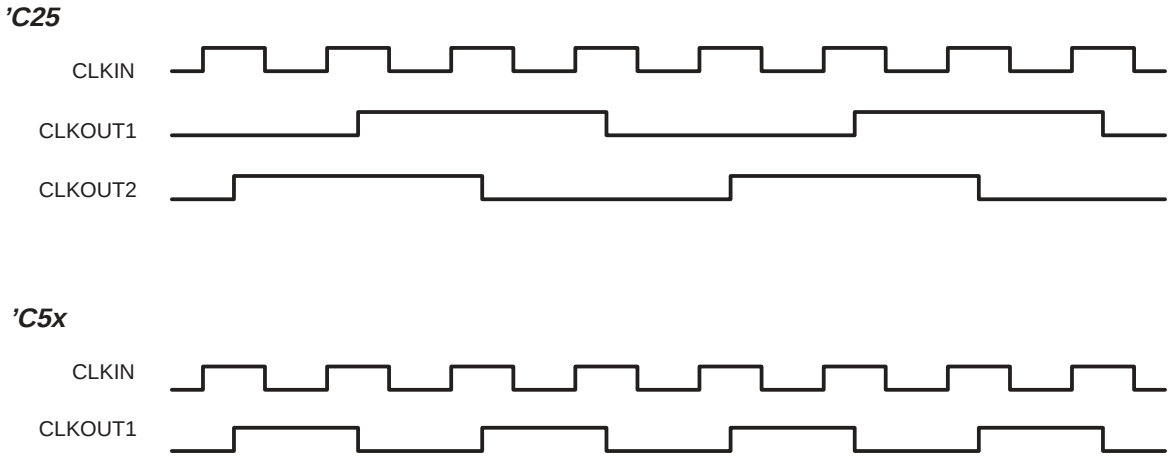
Some of the 'C25-equivalent pins have additional capabilities on the 'C5x. The 'C5x supports external direct memory access of the on-chip single-access RAM block. For this reason, the following signals are now bidirectional:

- Address lines, A0–A15
- Memory access strobe, $\overline{\text{STRB}}$
- Read/write, R/$\overline{\text{W}}$
- Bus request, $\overline{\text{BR}}$

The ’C5x serial port transmit clock (CLKX) can now be configured as an output that operates at one-fourth the machine clock rate. CLKX is configured as an input by reset. The ’C25 CLKX pin is always an input.

The ’C25 operates with a four-phase clock. The ’C25 machine rate is one-fourth the CLKIN rate. CLKOUT1 and CLKOUT2 operate at the machine rate and are 90° out of phase. The ’C5x operates with a two-phase clock. The ’C5x machine rate is one-half the CLKIN rate. In addition, the ’C5x offers a divide-by-one clock input feature so that the ’C5x machine rate equals the CLKIN rate. CLKOUT1 operates at the machine rate. Figure C–4 shows both the ’C25 and the ’C5x clocking schemes.

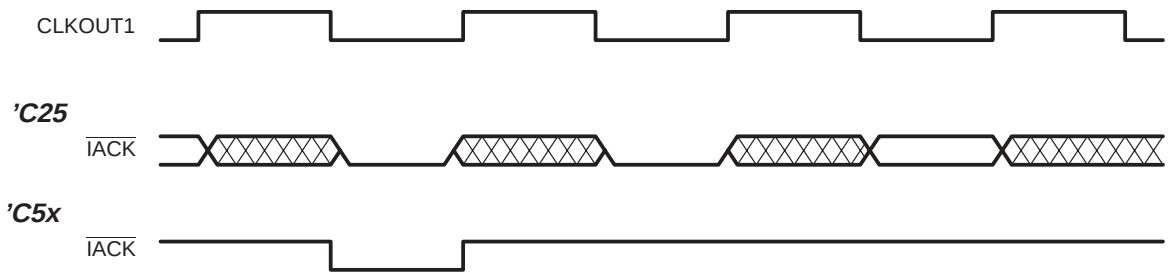
Figure C–4. TMS320C25 and TMS320C5x Clocking Schemes



The ’C5x MP/$\overline{\text{MC}}$ pin is sampled only while $\overline{\text{RS}}$ is low. Changes on this pin are ignored while $\overline{\text{RS}}$ is high. The mode can be changed during execution by changing the MP/$\overline{\text{MC}}$ bit in the PMST. On the ’C25, any change on the MP/$\overline{\text{MC}}$ pin affects the operation of the device, regardless of the state of $\overline{\text{RS}}$.

The ’C5x $\overline{\text{IACK}}$ signal goes low only on the first machine cycle of the fetch of the first word of the interrupt vector. The ’C25 $\overline{\text{IACK}}$ goes low on each wait-state cycle, as well as on the first machine cycle, but it is valid only during CLKOUT1 low (during CLKOUT1 high, it has a specific meaning for emulator/test operations). Figure C–5 illustrates this difference.

Figure C–5. TMS320C25 $\overline{IACK}$ Versus TMS320C5x $\overline{IACK}$



The ’C5x device includes some additional functions not included with the ’C25. These functions and associated pins are as follows:

- TDM serial port: TCLKR, TCLKX, TDR, TDX, TADD, TFRM
- Emulation interface: EMU0, EMU1/$\overline{OFF}$, $\overline{IAQ}$, TCK, TDI, TDO, TMS, TRST
- Timer borrow: TOUT
- Divide-by-one clock: CLKIN2, CLKMD1, and CLKMD2
- Fourth external interrupt: $\overline{INT4}$
- Nonmaskable interrupt: $\overline{NMI}$
- Read enable: $\overline{RD}$
- Write enable: $\overline{WE}$

The ’C5x package also includes 12 additional power and 13 additional ground pins. These additional power and ground pins enable the device to operate at much faster speeds. Twenty pins are reserved for future ’C5x spinoff devices.

C.2 Timing

The ’C2x and the ’C5x operate with some timing differences. These timing differences include aspects of the on-chip operation and the external memory interfacing. These key differences are:

- The ’C5x is capable of operating at two to three times the speed of a ’C2x.
- The ’C2x operates with a three-deep pipeline, while the ’C5x operates with a four-deep pipeline.
- The ’C5x external memory interface is faster and includes external interface enhancements.
- Some compatible operations execute in a different number of machine cycles.

C.2.1 Device Clock Speed

The ’C2x operates its machine cycles with a divide-by-four clocking scheme. The ’C5x uses a divide-by-two clocking scheme. This means that a ’C2x, operating with a 40-MHz CLKIN, executes its machine cycles within 100 ns, while the ’C5x, which is operating with the same CLKIN, executes its machine cycles in 50 ns. This clocking arrangement changes the way that the signals of the devices are specified. Many of the ’C2x timing values, given in the *TMS320 Second-Generation Digital Signal Processor Data Sheet*, are specified as quarter-phase (Q) + N ns. The timing values of the ’C5x are defined in half-phases (H).

C.2.2 Pipeline

The ’C2x operates with a three-deep pipeline, while the ’C5x operates with a four-deep pipeline. This means that anytime there is a program counter (PC) discontinuity (for example, branch, call, return, interrupt, etc.), it takes four cycles to complete with the ’C5x, whereas it takes three cycles on the ’C2x. The ’C5x, however, also has delayed instructions that take only two cycles to complete.

C.2.3 External Memory Interfacing

The ’C5x is designed to execute external memory operations with the same signals as the ’C2x. As mentioned above, the ’C5x operates at twice the instruction rate of the ’C2x when both operate with the same input clock. The ’C5x uses its software wait-state generators to compensate for this interface difference. The ’C5x device, operating with one software wait state, has similar memory timing to the ’C2x operating with no wait states. However, external writes require two cycles on the ’C5x devices. The exact timing of the signals differ because of the more advanced process used with the ’C5x.

The ’C5x has two additional memory interface signals to reduce the amount of external interfacing circuitries. The $\overline{RD}$ signal can be used to interface directly to the output enable pin of another device, while the $\overline{WE}$ signal can be directly connected to the write enable pin of another device. This alleviates the need of gating $\overline{STRB}$ and R/$\overline{W}$ to generate the equivalent signals.

C.2.4 Execution Cycle Times

Some of the ’C2x instructions require additional cycles or words to execute on the ’C5x. The function of these instructions is the same, but the format and pipeline execution are enhanced to operate with the ’C5x architecture.

The IN and OUT instructions are now two-word instructions. They execute on the ’C5x in the same number of cycles as with the ’C2x, but the assembler generates a two-word instruction for the ’C5x. Note that the ’C5x IN and OUT instructions behave differently in RPT mode. See Chapter 6, *Assembly Language Instructions*, for details. Two words are used because the ’C5x can address 65 536 I/O ports; the ’C2x only addresses 16. The ’C5x can address sixteen of its I/O ports in data memory space. This allows any instruction with data-memory-addressing capability to also read or write directly to an I/O port instead of having to pass it through a temporary on-chip data memory location. For example, a value can be read directly from an external analog-to-digital (A/D) converter into the ALU via an I/O port.

The modification of the three mode bits of the serial port are executed in two-cycle/two-word instructions with the ’C5x. However, any or all of the three bits can be modified with one instruction without affecting other bits in the register. This is done with the PLU instructions.

The NORM instruction modifies the auxiliary register (AR) on the execute (fourth) phase of the pipeline, while the ARAU operations occur on the decode (second) phase. The two instructions following a NORM instruction should not use the same AR for an address. If the two instructions following NORM change the auxiliary register pointer (ARP), then the NORM update of the AR is executed on the new ARP, not the old one. See Chapter 6, *Assembly Language Instructions*, for NORM instruction description. The assembler supports an optional way to test for this condition and automatically compensate by adding NOP instructions to the code. This modification is made to the listing and object files and does not affect your source code.

Unlike the ’C2x, the ARs are also accessible in the data address space on the ’C5x. This allows the ARs to be loaded with the CALU instructions for advanced-addressing modes. However, use caution when using this feature because the CALU operations write to the ARs on the execute phase of the pipeline and, therefore, are subject to the same characteristics of the NORM instruction. The assembler supports the option to flag these conflicts for resolution.

C.3 On-Chip Peripheral Interfacing

The ’C5x has more peripherals than the ’C2x; many ’C5x peripherals are enhancements of the ’C2x peripherals. The ’C2x has three peripheral circuits: serial port, timer, and 16 I/O ports. In addition to these peripherals, the ’C5x has software wait states and a divide-by-one clock.

The serial port interface of the ’C5x has been enhanced because the CLKX pin can be configured as either an input or an output. (CLKX is always an input on the ’C2x.) CLKX is configured as an input upon a device reset to maintain compatibility with the ’C2x. The new serial port status bits are now mapped to a memory-mapped register that is used exclusively for the serial port. The serial port modes are no longer controlled via status register 1 (ST1). Therefore, serial port modes changed by using the LST1 instruction will no longer work. The mode bits must be set/reset via the serial port control register (SPC). The data transmit (DXR) and data receive (DRR) registers have been moved in the memory map from locations 1 and 0, to 33 and 32, respectively. See Section 9.7, *Serial Port Interface*, on page 9-23 for more details.

The timer has been enhanced on the ’C5x to include a divide-down factor of 1 to 17 and can be stopped or reset via software. These additional features are controlled via the timer control register (TCR). Upon reset, the divide-down factor is set to 1, and the timer is enabled to maintain compatibility with the ’C2x. The timer (TIM) and period (PRD) registers have been moved in the memory map from locations 2 and 3, to locations 36 and 37, respectively. See Section 9.3, *Timer*, on page 9-9 for more details.

The 16 I/O ports of the ’C5x are addressable in the data memory space. Any instruction that can address data memory can also address the I/O ports. This allows direct access of the I/O space by the CPU and supports bit operation in the I/O space via the PLU. The I/O space can be increased from 16 ports to 65 536 ports. However, no additional decode circuitry is necessary if only 16 ports are used. See Section 8.5, *Input/Output (I/O) Space*, on page 8-22 and Section 9.6, *Parallel I/O Ports*, on page 9-22 for more details.

The ’C5x includes software wait-state generators that are mapped on 16K-word page sizes in the program and data memory spaces. There are also wait-state generators for the I/O ports. The I/O space wait-state generators can be mapped on 2-word or on 8K-word boundaries. These wait-state generators allow the system to be programmed for 0, 1, 2, 3, 4, or 7 wait states, eliminating the need of an off-chip interfacing circuitry. External access wait states can be extended further via the READY signal. See Section 9.4, *Software-Programmable Wait-State Generators*, on page 9-13 for more details.

C.4 'C2x-to-'C5x Instruction Set

The 'C5x instruction set is a superset of the 'C2x instruction set. The instruction set of the 'C2x is upward source-code compatible. This means that all of the instruction features of the 'C2x, implemented and code written for the 'C2x, can be reassembled to run on the 'C5x. See Chapter 6, *Assembly Language Instructions*, for the detailed discussion of the instruction set.

C.4.1 Overview

There are a number of new instructions on the 'C5x devices. These new instructions provide an advanced addressing scheme and exercise the new CPU enhancements. To simplify the description of the instruction set, a number of different instructions are combined into single new instructions with additional operand formats, such as the ADD instruction shown in Table C–1.

Table C–1. TMS320C2x Versus TMS320C5x for the ADD Instruction

'C2x Instruction	'C5x Instruction
ADD *+	ADD *+
ADDK 0FFh	ADD #0FFh
ADLK 0FFFFh	ADD #0FFFFh
ADDH *+	ADD *+ , 16

The IDLE instruction, when executed, stops the CPU from fetching and executing instructions until an unmasked interrupt occurs. The 'C2x automatically enables the interrupts globally with the execution of the IDLE instruction; this saves the extra instruction word/cycle required to execute the EINT (enable interrupts globally) instruction. Upon receipt of the interrupt, the 'C2x executes the interrupt vector and resumes operations.

The 'C5x does not automatically enable the interrupts globally with its IDLE instruction. If the interrupts are not globally enabled (INTM = 1), then the CPU resumes execution at the instruction following the IDLE instruction, without taking the interrupt trap. If the interrupts are globally enabled (INTM = 0), the 'C5x operates like the 'C2x. In addition, a second low-power mode is available with the IDLE2 instruction. This mode operates the same as IDLE, except that the CPU will resume only after an external interrupt. See Chapter 6, *Assembly Language Instructions*, for IDLE and IDLE2 instruction details.

The 'C5x repeat counter is 16 bits wide (the 'C2x repeat counter is 8 bits wide). This means that, when loading from RAM, the RPT instruction supports repeat counts up to 65 536. The assembler also allows the RPT to support a16-bit immediate repeat count. Note that RPT with long immediate addressing is, however, a two-word instruction.

C.4.2 Serial Port Control Bit Instructions

The serial port mode control bits have been moved from the status registers to the serial port control register (SPC). Because they are no longer part of the CPU registers, they no longer have direct instructions to set or clear them. The bits of the SPC can be manipulated easily with the PLU instructions (Table 6–6 on page 6-14). Table C–2 shows the 'C5x serial port instructions that replace the 'C2x instructions (note that the data page pointer must be set to 0 to execute these new instructions).

Table C–2. TMS320C2x to TMS320C5x Serial Port Instructions

'C2x Instruction	'C5x Instruction
FORT0	APL #0FFFBh, SPC
FORT1	OPL #4, SPC
RFSM	APL #0FFF7h, SPC
RTXM	APL #0FFDFh, SPC
SFSM	OPL #8, SPC
STXM	OPL #020h, SPC

Any or all three of the SPC bits can be set in one execution of the OPL instruction, while any or all three of the bits can be cleared with the APL instruction. The SPC bits can be toggled with the XPL instruction. See Chapter 6, *Assembly Language Instructions*, for instruction details.

C.4.3 'C2x-to-'C5x Instruction Set Mapping

The Texas Instruments 'C5x assembler accepts instruction mnemonics from either the 'C2x or the 'C5x instruction set. Because the 'C5x instruction set is a superset of the 'C2x instruction set, there are some 'C5x instructions that do not appear in the following tables. Table C–3 through Table C–8 alphabetically list the maps between the 'C2x and 'C5x instruction sets within the following functional groups:

- ❏ Accumulator memory reference instructions (Table C–3)
- ❏ Auxiliary registers and data memory page pointer instructions (Table C–4 on page C-15)
- ❏ TREG0, PREG, and multiply instructions (Table C–5 on page C-16)
- ❏ Branch and call instructions (Table C–6 on page C-17)
- ❏ I/O and data memory operation instructions (Table C–7 on page C-18)
- ❏ Control instructions (Table C–8 on page C-19)

Table C–3. TMS320C2x-to-TMS320C5x Accumulator Memory Reference Instructions

'C2x Instruction	'C5x Instruction
ABS	ABS
ADD	ADD
ADDC	ADDC
ADDH	ADD
ADDK	ADD
ADDS	ADDS
ADDT	ADDT
ADLK	ADD
AND	AND
ANDK	AND
CMPL	CMPL
LAC	LACC
LACK	LACL
LACT	LACT
LALK	LACC
NEG	NEG
NORM	NORM†
OR	OR
ORK	OR
ROL	ROL
ROR	ROR
SACH	SACH
SACL	SACL
SBLK	SUBB
SFL	SFL
SFR	SFR
SUB	SUB
SUBB	SUBB

† There is a potential pipeline conflict with the NORM instruction. See the NORM instruction summary (page 6-181) for details.

Table C–3. TMS320C2x-to-TMS320C5x Accumulator Memory Reference Instructions (Continued)

'C2x Instruction	'C5x Instruction
SUBC	SUBC
SUBH	SUB
SUBK	SUB
SUBS	SUBS
SUBT	SUBT
XOR	XOR
XORK	XOR
ZAC	LACL
ZALH	LACC
ZALR	ZALR
ZALS	LACL

Table C–4. TMS320C2x-to-TMS320C5x Auxiliary Registers and Data Memory Page Pointer Instructions

'C2x Instruction	'C5x Instruction
ADRK	ADRK
CMPR	CMPR
LAR	LAR
LARK	LAR
LARP	MAR
LDP	LDP
LDPK	LDP
LRLK	LAR
MAR	MAR
SAR	SAR
SBRK	SBRK

Table C–5. TMS320C2x-to-TMS320C5x TREG0, PREG, and Multiply Instructions

'C2x Instruction	'C5x Instruction
APAC	APAC
LPH	LPH
LT	LT
LTA	LTA
LTD	LTD
LTP	LTP
LTS	LTS
MAC	MAC
MACD	MACD
MPY	MPY
MPYA	MPYA
MPYK	MPY
MPYS	MPYS
MPYU	MPYU
PAC	PAC
SPAC	SPAC
SPH	SPH
SPL	SPL
SPM	SPM
SQRA	SQRA
SQRS	SQRS

Table C–6. TMS320C2x-to-TMS320C5x Branch and Call Instructions

'C2x Instruction	'C5x Instruction
B	B
BACC	BACC
BANZ	BANZ
BBNZ	BCND
BBZ	BCND
BC	BCND
BGEZ	BCND
BGZ	BCND
BIOZ	BCND
BLEZ	BCND
BLZ	BCND
BNC	BCND
BNV	BCND
BNZ	BCND
BV	BCND
BZ	BCND
CALA	CALA
CALL	CALL
RET	RET
TRAP	TRAP

Table C–7. TMS320C2x-to-TMS320C5x I/O and Data Memory Operation Instructions

'C2x Instruction	'C5x Instruction
BLKD	BLDD
BLKP	BLPD
DMOV	DMOV
FORT†	OPL APL
IN	IN
OUT	OUT
RFSM†	APL
RTXM†	APL
RXF	CLRC
SFSM†	OPL
STXM	OPL
SXF	SETC
TBLR	TBLR
TBLW	TBLW

† The suggested mapping requires that the data page pointer be set to 0.

Table C–8. TMS320C2x-to-TMS320C5x Control Instructions

'C2x Instruction	'C5x Instruction
BIT	BIT
BITT	BITT
CNFD	CLRC
CNFP	SETC
DINT	SETC
EINT	CLRC
IDLE	IDLE
LST	LST
LST1	LST
NOP	NOP
POP	POP
POPD	POPD
PSHD	PSHD
PUSH	PUSH
RC	CLRC
RHM	CLRC
ROVM	CLRC
RPT	RPT
RPTK	RPT
RSXM	CLRC
RTC	CLRC
SC	SETC
SHM	SETC
SOVM	SETC
SST	SST
SST1	SST
SSXM	SETC
STC	SETC

Appendix D

Design Considerations for Using XDS510 Emulator

The ’C5x DSPs support emulation through a dedicated emulation port. The emulation port is a superset of the IEEE JTAG standard 1149.1 and can be accessed by the XDS510 emulator. The information in this appendix supports XDS510 Cable #2563988-001 Rev B.

The term *JTAG,* as used in this book, refers to TI scan-based emulation, which is based on the IEEE standard 1149.1.

For more information concerning the IEEE standard 1149.1, contact IEEE Customer Service:

Address: IEEE Customer Service
445 Hoes Lane, PO Box 1331
Piscataway, NJ 08855-1331

Phone: (800) 678–IEEE in the US and Canada
(908) 981–1393 outside the US and Canada

FAX: (908) 981–9667 Telex: 833233

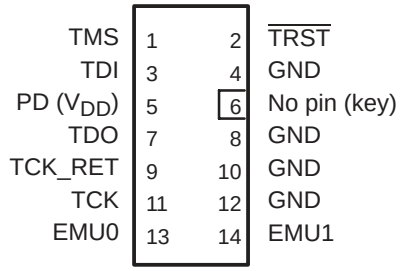
Topic **Page**

D.1 Cable Header and Signals

To perform emulation with the XDS510, your target system must have a 14-pin header (two 7-pin rows) with connections as shown in Figure D–1. Table D–1 describes the emulation signals. Although you can use other headers, recommended parts include:

Straight header, unshrouded	DuPont Electronics® part number 67996–114
Right-angle header, unshrouded	DuPont Electronics® part number 68405–114

Figure D–1. Header Signals and Header Dimensions



Header Dimensions:
Pin-to-pin spacing: 0.100 in. (X,Y)
Pin width: 0.025 in. square post
Pin length: 0.235 in., nominal

Table D–1. XDS510 Header Signal Description

Pin	Signal	State	Target State	Description
1	TMS	O	I	JTAG test mode select
2	$\overline{\text{TRST}}$	O	I	JTAG test reset
3	TDI	O	I	JTAG test data input
5	PD	I	O	Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD should be tied to V_{DD} in the target system.
7	TDO	I	O	JTAG test data output
9	TCK_RET	I	O	JTAG test clock return. Test clock input to the XDS510 emulator. May be a buffered or unbuffered version of TCK.
11	TCK	O	I	JTAG test clock. TCK is a 10-MHz clock source from the emulation cable pod. This signal can be used to drive the system test clock.
13	EMU0	I	I/O	Emulation pin 0
14	EMU1	I	I/O	Emulation pin 1

D.2 Bus Protocol

The IEEE standard 1149.1 covers the requirements for JTAG bus slave devices (’C5x) and provides certain rules, summarized as follows:

- The TMS and TDI inputs are sampled on the rising edge of the TCK signal of the device.
- The TDO output is clocked from the falling edge of the TCK signal of the device.

When JTAG devices are daisy-chained together, the TDO of one device has approximately a half TCK cycle setup time before the next device’s TDI signal. This timing scheme minimizes race conditions that would occur if both TDO and TDI were timed from the same TCK edge. The penalty for this timing scheme is a reduced TCK frequency.

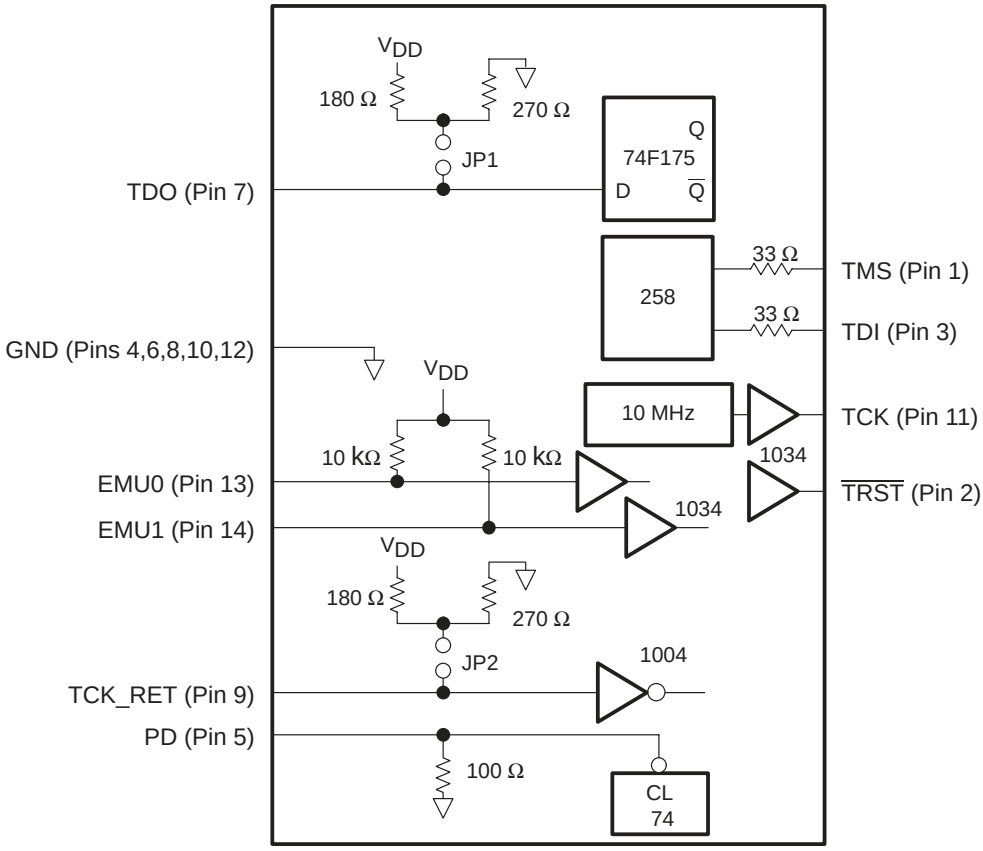
The IEEE standard 1149.1 does not provide rules for JTAG bus master (XDS510) devices. Instead, it states that it expects a bus master to provide bus slave compatible timings. The XDS510 provides timings that meet the bus slave rules and also provides an optional timing mode that allows you to run the emulation at a much higher frequency for improved performance.

D.3 Emulator Cable Pod

Figure D–2 shows a portion of the XDS510 emulator cable pod. The functional features of the emulator pod are:

- TDO and TCK_RET can be parallel-terminated inside the pod if required by the application. By default, these signals are not terminated.
- TCK is driven with a 74AS1034 device. Because of the high-current drive (48 mA I_{OL}/I_{OH}), this signal can be parallel-terminated. If TCK is tied to TCK_RET, you can use the parallel terminator in the pod.
- TMS and TDI can be generated from the falling edge of TCK_RET, according to the IEEE (JTAG) standard 1149.1 bus slave device timing rules. They can also be driven from the rising edge of TCK_RET, which allows a higher TCK_RET frequency. The default is to match the IEEE standard 1149.1 slave device timing rules. This is an emulator software option that can be selected when the emulator is invoked. In general, single-processor applications can benefit from the higher clock frequency. However, in multiprocessing applications, you may wish to use the IEEE standard 1149.1 bus slave timing mode to minimize emulation system timing constraints.
- TMS and TDI are series-terminated to reduce signal reflections.
- A 10-MHz test clock source is provided. You can also provide your own test clock for greater flexibility.

Figure D–2. Emulator Cable Pod Interface



NOTE:
All devices are 74AS, unless otherwise specified.

D.4 Emulator Cable Pod Signal Timings

Figure D–3 shows the signal timings for the emulator cable pod. Table D–2 defines the timing parameters illustrated in the figure. These timing parameters are calculated from values specified in the standard data sheets for the cable pod and are for reference only. Texas Instruments does not test or guarantee these timings.

The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

Figure D–3. Emulator Cable Pod Timings

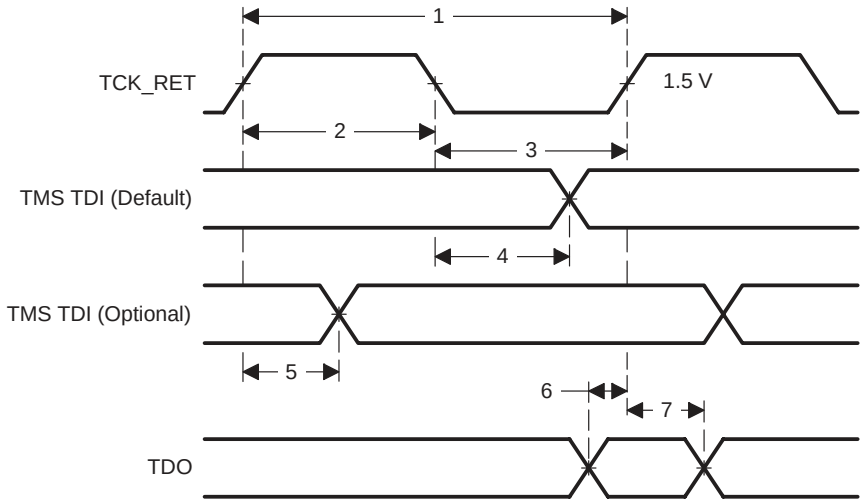


Table D–2. Emulator Cable Pod Timing Parameters

No.	Paramter	Description	Min	Max	Unit
1	t_{TCKmin} t_{TCKmax}	TCK_RET period	35	200	ns
2	$t_{TCKhighmin}$	TCK_RET high pulse duration	15		ns
3	$t_{TCKlowmin}$	TCK_RET low pulse duration	15		ns
4	$t_{d(XTMXmin)}$ $t_{d(XTMXmax)}$	TMS/TDI valid from TCK_RET low (default timing)	6	20	ns
5	$t_{d(XTMSmin)}$ $t_{d(XTMSmax)}$	TMS/TDI valid from TCK_RET high (optional timing)	7	24	ns
6	$t_{su(XTDOmin)}$	TDO setup time to TCK_RET high	3		ns
7	$t_{hd(XTDOmin)}$	TDO hold time from TCK_RET high	12		ns

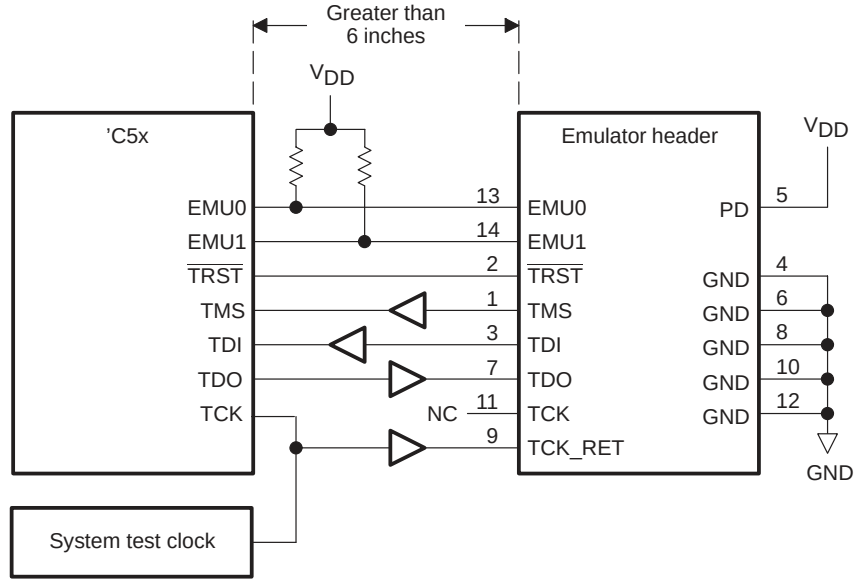
D.5 Target System Test Clock

Figure D–4 shows an application with the system test clock generated in the target system. In this application the TCK signal is left unconnected. There are two benefits to having the target system generate the test clock:

1) You can set the test clock frequency to match your system requirements. The emulator provides only a single 10-MHz test clock.

2) You may have other devices in your system that require a test clock when the emulator is not connected.

Figure D–4. Target-System Generated Test Clock

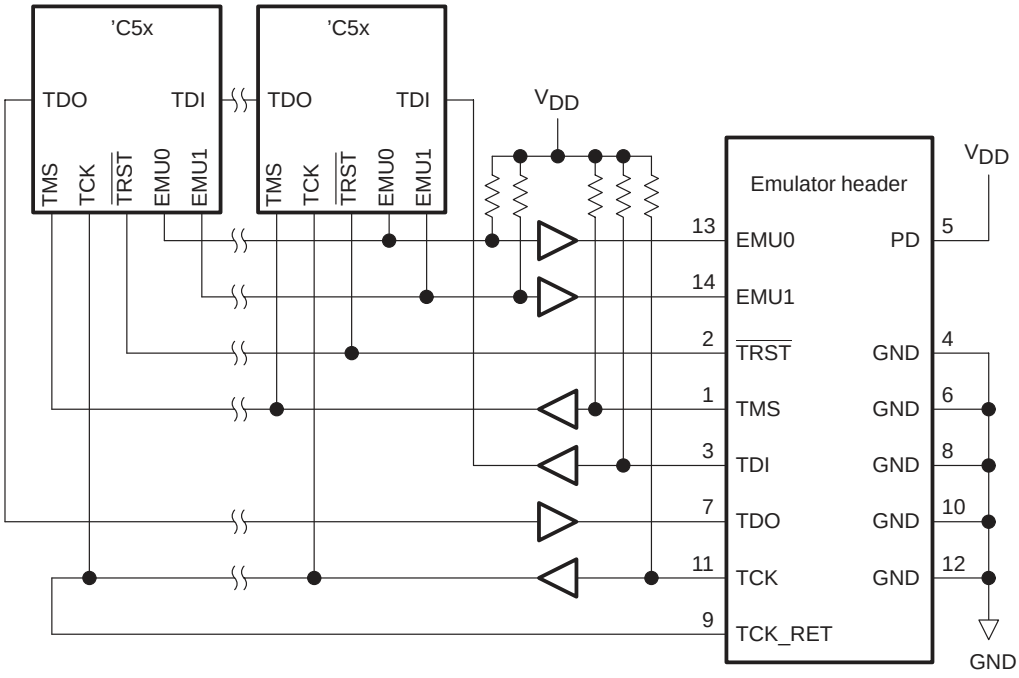


D.6 Configuring Multiple Processors

Figure D–5 shows a typical daisy-chained multiprocessor configuration that meets the minimum requirements of the IEEE (JTAG) standard 1149.1. The emulation signals are buffered to isolate the processors from the emulator and provide adequate signal drive for the target system. One of the benefits of this test interface is that you can slow down the test clock to eliminate timing problems. Several key points to multiprocessor support are as follows:

- The processor TMS, TDI, TDO, and TCK signals should be buffered through the same physical device package for better control of timing skew.
- The input buffers for TMS, TDI, and TCK should have pullup resistors connected to 5 volts to hold these signals at a known value when the emulator is not connected. A pullup resistor value of 4.7 kΩ or greater is suggested.
- Buffering EMU0 and EMU1 is optional but highly recommended to provide isolation. These are not critical signals and do not have to be buffered through the same physical package as TMS, TCK, TDI, and TDO.

Figure D–5. Multiprocessor Connections



D.7 Connections Between the Emulator and the Target System

It is extremely important to provide high-quality signals between the emulator and the target system. You must supply the correct signal buffering, test clock inputs, and multiple processor interconnections to ensure proper emulator and target system operation.

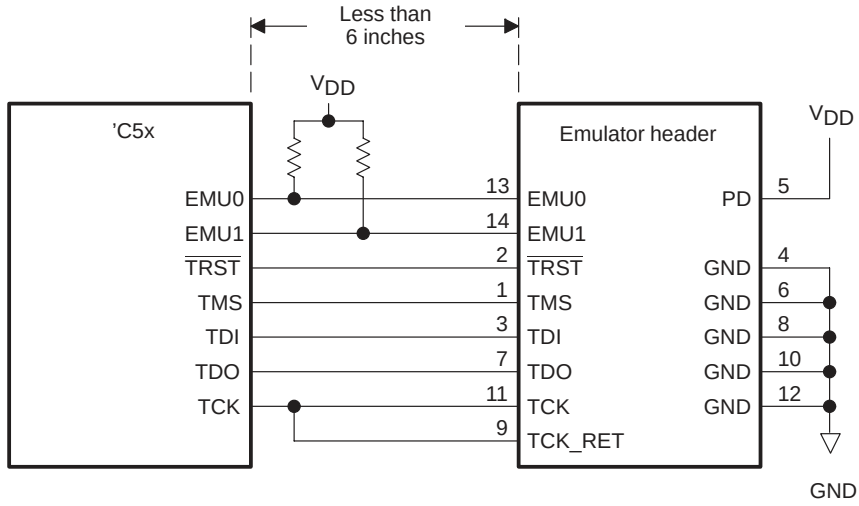
EMU0 and EMU1 are I/O pins on the ’C5x; however, they are only inputs to the XDS510. In general, these pins are used in multiprocessor systems to provide global run/stop operations.

D.7.1 Emulation Signals Not Buffered

If the distance between the emulation header and the target device is less than 6 inches, no buffering is necessary. Figure D–6 shows the no-buffering configuration.

The EMU0 and EMU1 signals must have pullup resistors connected to 5 volts to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

Figure D–6. Emulator Connections Without Signal Buffering



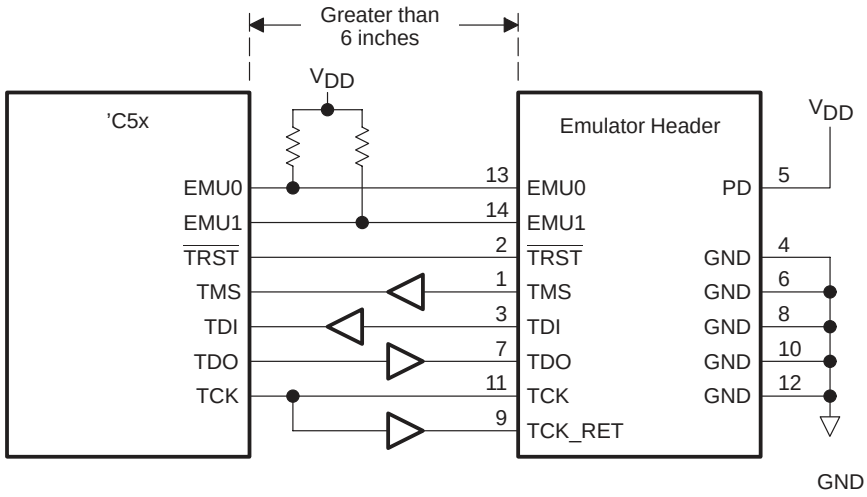
D.7.2 Emulation Signals Buffered

If the distance between the emulation header and the JTAG target device is greater than 6 inches, the emulation signals must be buffered. Figure D–7 shows the buffering configuration. Emulation signals TMS, TDI, TDO, and TCK_RET are buffered through the same device package.

The EMU0 and EMU1 signals must have pullup resistors connected to 5 volts to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

To have high-quality signals (especially the processor TCK and the emulator TCK_RET signals), you may have to employ special care when routing the printed wiring board trace. You also may have to use termination resistors to match the trace impedance. The emulator pod provides optional internal parallel terminators on the TCK_RET and TDO. TMS and TDI provide fixed series termination.

Figure D–7. Buffered Signals



D.8 Emulation Timing Calculations

The following are a few examples of how to calculate the emulation timings in your system. For actual target timing parameters, see the appropriate device data sheets.

Assumptions:

$t_{su(TTMS)}$	Target TMS/TDI setup to TCK high	10 ns
$t_{h(TTMS)}$	Target TMS/TDI hold from TCK high	5 ns
$t_{d(TTDO)}$	Target TDO delay from TCK low	15 ns
$t_{d(bufmax)}$	Target buffer delay maximum	10 ns
$t_{d(bufmin)}$	Target buffer delay minimum	1 ns
$t_{(bufskew)}$	Target buffer skew between two devices in the same package: $[t_{d(bufmax)} - t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{tckfactor}$	A 40/60 duty cycle clock	0.4

Given in Table D–2 (page D-6):

$t_{d(XTMSmax)}$	XDS510 TMS/TDI delay from TCK_RET low, maximum	20 ns
$t_{d(XTMX)}$	min XDS510 TMS/TDI delay from TCK_RET low, minimum	6 ns
$t_{d(XTMSmax)}$	XDS510 TMS/TDI delay from TCK_RET high, maximum	24 ns
$t_{d(XTMXmin)}$	XDS510 TMS/TDI delay from TCK_RET high, minimum	7 ns
$t_{su(XTDOmin)}$	TDO setup time to XDS510 TCK_RET high	3 ns

There are two key timing paths to consider in the emulation design:

1) the TCK_RET/TMS/TDI (t_{prdtck_TMS}) path, and

2) the TCK_RET/TDO (t_{prdtck_TDO}) path.

In each case, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Case 1: Single processor, direct connection, TMS/TDI timed from TCK_RET low (default timing).

$t_{prdtck\_TMS} = [t_{(d(XTMSmax)} + t_{su(TTMS)}] / t_{tckfactor}$
$= (20 ns + 10 ns) / 0 .4$
$= 75 ns (13.3 MHz)$

$t_{prdtck\_TDO} = [t_{(d(TTDO)} + t_{su(XTDOmin)}] / t_{tckfactor}$
$= (15 ns + 3 ns) / 0.4$
$= 45 ns (22.2 MHz)$

In Case 1, the TCK/TMS path is the limiting factor.

Case 2: Single processor, direct connection, TMS/TDI timed from TCK_RET high (optional timing).

$t_{prdtck\_TMS} = t_{d(XTMSmax)} + t_{su(TTMS)}$
$= (24 ns + 10 ns)$
$= 34 ns (29.4 MHz)$

$t_{prdtck\_TDO} = [t_{d(TTDO)} + t_{su(XTDOmin)}] / t_{tckfactor}$
$= (15 + 3) / 0.4$
$= 45 ns (22.2 MHz)$

In Case 2, the TCK/TDO path is the limiting factor. One other thing to consider in this case is the TMS/TDI hold time. The minimum hold time for the XDS510 cable pod is 7 ns, which meets the 5-ns hold time of the target device.

Case 3: Single/multiple processor, TMS/TDI buffered input; TCK_RET/TDO buffered output, TMS/TDI timed from TCK_RET high (optional timing).

$t_{prdtck\_TMS} = t_{d(XTMSmax)} + t_{su(TTMS)} + 2t_{d(bufmax)}$
$= 24 ns + 10 ns + 2 (10)$
$= 54 ns (18.5 MHz)$

$t_{prdtck\_TDO} = [t_{d(TTDO)} + t_{su(XTDOmin)} + t_{(bufskew)}] / t_{tckfactor}$
$= (15 ns + 3 ns + 1.35 ns) / 0.4$
$= 58.4 ns \ (20.7 MHz)$

In Case 3, the TCK/TMS path is the limiting factor. The hold time on TMS/TDI is also reduced by the buffer skew (1.35 ns) but still meets the minimum device hold time.

Case 4: Single/multiprocessor, TMS/TDI/TCK buffered input; TDO buffered output, TMS/TDI timed from TCK_RET low (default timing).

t_{prdtck_TMS} = $[t_{d(XTMSmax)} + t_{su(TTMS)} + t_{bufskew}] / t_{ckfactor}$
= (24 ns + 10 ns + 1.35 ns) / 0.4
= 88.4 ns (11.3 MHz)

t_{prdtck_TDO} = $[t_{d(TTDO)} + t_{su(XTDOmin)} + t_{d(bufmax)}] / t_{ckfactor}$
= (15 ns + 3 ns + 10 ns) / 0.4
= 70 ns (14.3 MHz)

In Case 4, the TCK/TMS path is the limiting factor.

In a multiprocessor application, it is necessary to ensure that the EMU0 and EMU1 lines can go from a logic low level to a logic high level in less than 10 µs. This can be calculated as follows (remember that t = 5 RC):

t_{rise} = $5(R_{pullup} \times N_{devices} \times C_{load\_per\_device})$
= $5(4.7k\Omega \times 16 \times 15pF)$
= 5.64 µs

Appendix E

Memories, Sockets, and Crystals

This appendix provides product information regarding memories and sockets that are manufactured by Texas Instruments and are compatible with the ’C5x. Information is also given regarding crystal frequencies, specifications, and vendors.

Topic **Page**

E.1 Memories

This section provides product information on EPROM memories that can be interfaced with ’C5x processors. Refer to *Digital Signal Processing Applications with the TMS320 Family* for additional information on interfaces using memories and analog conversion devices.

Data sheets for EPROM memories are located in the *MOS Memory Data Book* (literature number SMYD095):

TMS27C64
TMS27C128
TMS27C256
TMS27C512

Another EPROM memory, TMS27C291/292, is described in a data sheet (literature number SMLS291).

E.2 Sockets

AMP manufactures a 132-pin quad flat pack socket for the ’C5x devices. There are two pieces — a base (the socket itself) and a lid. The part numbers are:

- Base: AMP part number 821942-1
- Lid: AMP part number 821949-5

For additional information about TI sockets, contact the nearest TI sales office or:

Texas Instruments Incorporated
Connector Systems Dept, M/S 14–3
Attleboro, MA 02703
(617) 699–5242/5269
Telex: 92–7708

E.3 Crystals

This section lists the commonly used crystal frequencies (Table E–1), crystal specification requirements, and the names of suitable vendors.

Table E–1. Commonly Used Crystal Frequencies

Device	Frequency
TMS320C25	40.96 MHz
TMS320C5x	20.48 MHz 40.96 MHz

When connected across X1 and X2/CLKIN of the TMS320 processor, a crystal enables the internal oscillator. Crystal specification requirements are listed below:

Load capacitance = 20 pF
Series resistance = 30 ohm
Power dissipation = 1 mW

Vendors of crystals suitable for use with TMS320 devices are listed below:

RXD, Inc.
Norfolk, NB
(800) 228–8108

N.E.L. Frequency Controls, Inc.
Burlington, WI
(414) 763–3591

CTS Knight, Inc.
Contact the local distributor.

Appendix F

Submitting ROM Codes to TI

Texas Instruments offers a mask-programmable ROM to provide a single-chip solution to its customers. This appendix explains the benefits of the space-saving ROM and describes the function of the TMS320 development tools.

Topic **Page**

F.1 Single-Chip Solution

The size of a printed circuit board is a consideration in many DSP applications. To make full use of the board space, Texas Instruments offers this ROM code option that reduces the chip count and provides a single-chip solution. This option allows you to use a code-customized processor for a specific application while taking advantage of:

- ❏ Greater memory expansion
- ❏ Lower system cost
- ❏ Less hardware and wiring
- ❏ Smaller PCB

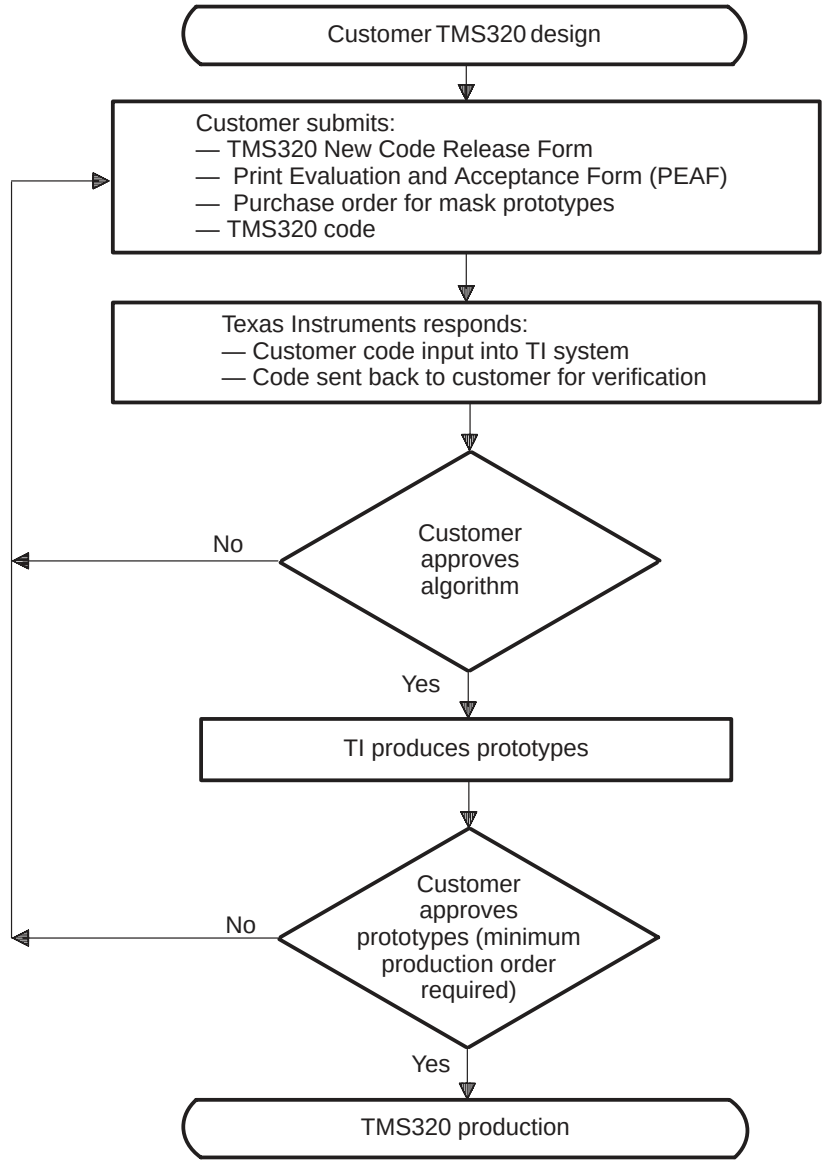
If a routine or algorithm is used often, it can be programmed into the on-chip ROM of a TMS320 DSP. TMS320 programs can also be expanded by using external memory; this reduces chip count and allows for a more flexible program memory. Multiple functions are easily implemented by a single device, thus enhancing system capabilities.

TMS320 development tools are used to develop, test, refine, and finalize the algorithms. The microprocessor/microcomputer (MP/$\overline{\text{MC}}$) mode is available on all ROM-coded TMS320 DSP devices when accesses to either on-chip or off-chip memory are required. The microprocessor mode is used to develop, test, and refine a system application. In this mode of operation, the TMS320 acts as a standard microprocessor by using external program memory. When the algorithm has been finalized, the code can be submitted to Texas Instruments for masking into the on-chip program ROM. At that time, the TMS320 becomes a microcomputer that executes customized programs from the on-chip ROM. Should the code need changing or upgrading, the TMS320 can once again be used in the microprocessor mode. This shortens the field-upgrade time and avoids the possibility of inventory obsolescence.

F.2 TMS320 Development Flow

Figure F–1 illustrates the procedural flow for developing and ordering TMS320 masked parts. When ordering, there is a one-time, nonrefundable charge for mask tooling. A minimum production order per year is required for any masked-ROM device. ROM codes are deleted from the Texas Instruments system one year after the final delivery.

Figure F–1. TMS320 ROM Code Submittal Flowchart



F.3 Submitting TMS320 ROM Code

The TMS320 ROM code may be submitted in one of the following forms:

- 3-1/2-inch floppy: COFF format from macro-assembler/linker (preferred)
- 5-1/4-inch floppy: COFF format from macro-assembler/linker
- Modem (BBS): COFF format from macro-assembler/linker
- EPROM (others): TMS27C64
- PROM: TBP28S166, TBP28S86

When code is submitted to TI for masking, the code is reformatted to accommodate the TI mask-generation system. System-level verification by the customer is therefore necessary to ensure the reformatting remains transparent and does not affect the execution of the algorithm. The formatting changes involve the removal of address-relocation information (the code address begins at the base address of the ROM in the TMS320 device and progresses without gaps to the last address of the ROM) and the addition of data in the reserved locations of the ROM for device ROM test. Because these changes have been made, a checksum comparison is not a valid means of verification.

With each masked-device order, the customer must sign a disclaimer that states:

> The units to be shipped against this order were assembled, for expediency purposes, on a prototype (that is, nonproduction qualified) manufacturing line, the reliability of which is not fully characterized. Therefore, the anticipated inherent reliability of these prototype units cannot be expressly defined.

and a release that states:

> Any masked ROM device may be resymbolized as TI standard product and resold as though it were an unprogrammed version of the device, at the convenience of Texas Instruments.

The use of the ROM-protect feature does not hold for this release statement. Additional risk and charges are involved when the ROM-protect feature is selected. Contact the nearest TI Field Sales Office for more information on procedures, leadtimes, and cost associated with the ROM-protect feature.

Appendix G

Development Support and Part Order Information

This appendix provides development support information, device part numbers, and support tool ordering information for the ’C5x.

Each ’C5x support product is described in the *TMS320 DSP Development Support Reference Guide*. In addition, more than 100 third-party developers offer products that support the TI TMS320 family. For more information, refer to the *TMS320 Third-Party Support Reference Guide*.

For information on pricing and availability, contact the nearest TI Field Sales Office or authorized distributor. See the list at the back of this book.

Topic **Page**

G.1 Development Support

This section describes the development support provided by Texas Instruments.

G.1.1 Software and Hardware Development Tools

TI offers an extensive line of development tools for the ’C5x generation of DSPs, including tools to evaluate the performance of the processors, generate code, develop algorithm implementations, and fully integrate and debug software and hardware modules. The following products support development of ’C5x-based applications:

- Software Development Tools:
  - Assembler/linker
  - Simulator
  - Optimizing ANSI C compiler
  - Application algorithms
  - C/Assembly debugger and code profiler
- Hardware Development Tools:
  - Emulator XDS510
  - ’C5x Evaluation Module (EVM)
  - ’C5x DSP Starter Kit (DSK)

G.1.2 Third-Party Support

The TMS320 family is supported by products and services from more than 100 independent third-party vendors and consultants. These support products take various forms (both as software and hardware), from cross-assemblers, simulators, and DSP utility packages to logic analyzers and emulators. The expertise of those involved in support services ranges from speech encoding and vector quantization to software/hardware design and system analysis.

To ask about third-party services, products, applications, and algorithm development packages, contact the third party directly. Refer to the *TMS320 Third-Party Support Reference Guide* for addresses and phone numbers.

G.1.3 Technical Training Organization (TTO) TMS320 Workshops

TMS320C5x DSP Design Workshop. This workshop is tailored for hardware and software design engineers and decision-makers who will be designing and utilizing the ’C5x generation of DSP devices. Hands-on exercises throughout the course give participants a rapid start in developing ’C5x design skills. Microprocessor/assembly language experience is required. Experience with digital design techniques and C language programming experience is desirable.

These topics are covered in the ’C5x workshop:

- ❏ DSP fundamentals
- ❏ ’C5x architecture/instruction set
- ❏ Use of the PC-based software simulator
- ❏ Use of the ’C5x assembler/linker
- ❏ C programming environment
- ❏ System architecture considerations
- ❏ Memory and I/O interfacing
- ❏ Serial ports and multiple processor features

For registration information, pricing, or to enroll, call (972)644–5580.

G.1.4 Assistance

For assistance to TMS320 questions on device problems, development tools, documentation, software upgrades, and new products, you can contact TI. See *If You Need Assistance* in *Preface* for information.

G.2 Part Order Information

This section describes the part numbers of 'C5x devices, development support hardware, and software tools.

G.2.1 Device and Development Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all TMS320 devices and support tools. Each TMS320 member has one of three prefix designators: TMX, TMP, or TMS. Each support tool has one of two possible prefix designators: TMDX or TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (TMX/TMDX) through fully qualified production devices and tools (TMS/TMDS). This development flow is defined below.

Device Development Evolutionary Flow:

TMX The part is an experimental device that is not necessarily representative of the final device's electrical specifications.

TMP The part is a device from a final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification.

TMS The part is a fully qualified production device.

Support Tool Development Evolutionary Flow:

TMDX The development-support product that has not yet completed Texas Instruments internal qualification testing.

TMDS The development-support product is a fully qualified development support product.

TMX and TMP devices, and TMDX development-support tools are shipped with the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

TMS devices and TMDS development-support tools have been fully characterized, and the quality and reliability of the device has been fully demonstrated. Texas Instruments standard warranty applies to these products.

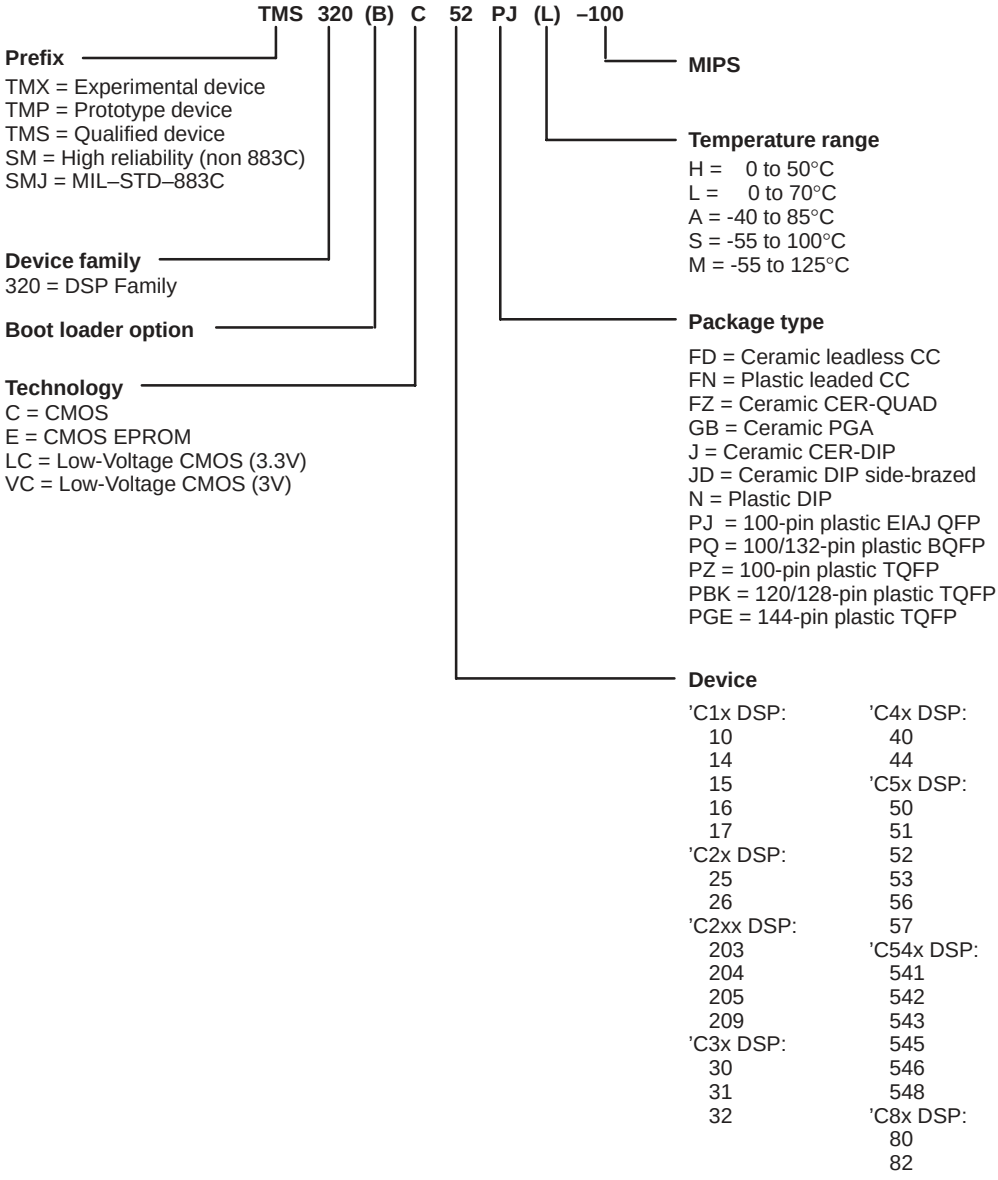
Note:

It is expected that prototype devices (TMX or TMP) have a greater failure rate than standard production devices. Texas Instruments recommends that these devices *not* be used in any production system, because their expected end-use failure rate is still undefined. Only qualified production devices should be used.

G.2.2 Device Nomenclature

TI device nomenclature includes the device family name and a suffix. Figure G–1 provides a legend for reading the complete device name for any TMS320 family member.

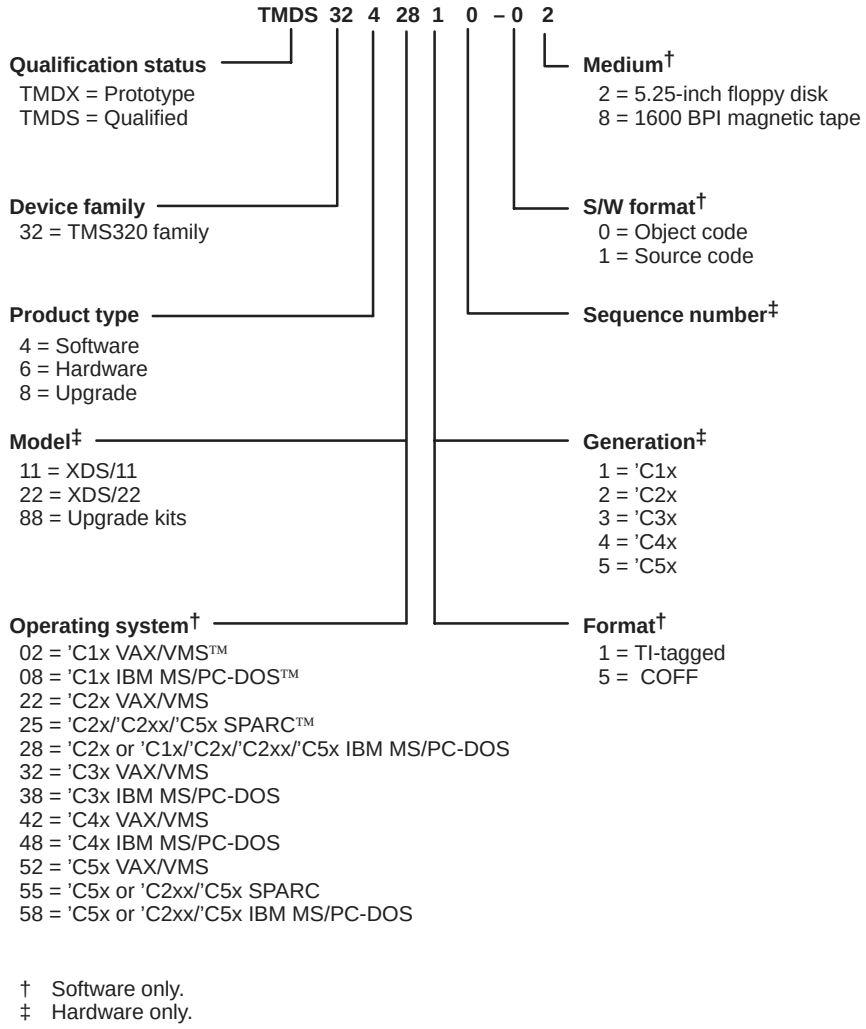
Figure G–1. TMS320 Device Nomenclature



G.2.3 Development Support Tools

Figure G–2 provides a legend for reading the part number for any TMS320 hardware or software development tool. Table G–1 lists the development support tools available for the ’C5x, the platform on which they run, and their part numbers.

Figure G–2. TMS320 Development Tool Nomenclature



† Software only.
‡ Hardware only.

Table G–1. TMS320C5x Development Support Tools Part Numbers

Development Tool	Platform	Part Number
Assembler/Linker	PC (DOS™, OS/2™)	TMDS3242850-02
C Compiler/Assembler/Linker	PC (DOS™, OS/2™)	TMDS3242855-02
C Compiler/Assembler/Linker	HP (HP-UX™) / SPARC™ (Sun OS™)	TMDS3242555-08
Debugger/Emulation Software	PC (DOS™, Windows™, OS/2™)	TMDS3240150
Debugger/Emulation Software	SPARC™ (Sun OS™)	TMDS3240650
Digital Filter Design Package	PC (DOS™)	DFDP
DSP Starter Kit (DSK)	PC (DOS™)	TMDS3200051
Evaluation Module (EVM)	PC (DOS™, Windows™)	TMDS3260050
Simulator (C language)	PC (DOS™, Windows™)	TMDS3245851-02
Simulator (C language)	SPARC™ (Sun OS™)	TMDS3245551-09
XDS510XL Emulator†	PC (DOS™, OS/2™)	TMDS00510
XDS510WS Emulator‡	SPARC™ (Sun OS™)	TMDS00510WS
3 V/5 V PC/SPARC JTAG Emulation Cable	XDS510™ / XDS510WS™	TMDS3080002

† Includes XDS510 board and JTAG cable
‡ Includes XDS510WS box and JTAG cable

G.3 Hewlett-Packard E2442A Preprocessor 'C5x Interface

The Hewlett-Packard E2442A preprocessor 'C5x interface provides a mechanical and electrical connection between your target system and an HP logic analyzer. Preprocessor hardware captures processor signals and passes them to the logic analyzer at the appropriate time, depending on the type of measurement you are making. With the preprocessor plugged in, both state and timing analysis is available. Two connectors are loaded onto the preprocessor to facilitate communications with other debugging tools. A BNC connector, when used with the sequencer of the logic analyzer halts the processor on a condition. Then you can use the 'C5x HLL debugger to examine the state of the system (for example, microprocessor registers). Likewise, a 14-pin connector is available to receive signals from the XDS510 development system. These signals can be used when defining a trigger condition for the analyzer.

The preprocessor includes software which automatically labels address, data, and status lines. Additionally, a disassembler is included. The disassembler processes state traces and displays the information on TMS320 mnemonics.

G.3.1 Capabilities

The preprocessor supports three modes of operation: in the first mode, *State per Transfer,* the preprocessor clocks the logic analyzer only when a bus transfer is complete. In this mode, wait and halt states are filtered out. In the second mode, CLKOUT1 clocks the logic analyzer every time the microprocessor is clocked. This mode captures all bus states. An example application would be to locate memory locations that do not respond to requests for data. In the third mode, you can use the preprocessor to make timing measurements.

The JTAG TAP (test access port) controller can be monitored in realtime. TAP state can be viewed under the predefined label *TAP.*

G.3.2 Logic Analyzers Supported

The preprocessor 'C5x interface supports the following logic analyzers:

- ❑ HP 1650A/B
- ❑ HP 16510B
- ❑ HP 16511B
- ❑ HP 16540/41(A/D)
- ❑ HP 16550A
- ❑ HP 1660A/61A/62A

G.3.3 Pods Required

There are eight pod-connectors on the preprocessor. Three are terminated and best used for state analysis as all signals needed for disassembly are available. The other five connectors are not terminated and contain all processor signals, including a second set of the signals needed for disassembly. This allows you to double probe these signals, making simultaneous state and timing measurements.

G.3.4 Termination Adapters (TAs)

Of the eight pods, three are terminated. You may need to order up to five termination adapters, depending on how many pods are connected at the same time.

G.3.5 Availability

For more information and availability of the Hewlett-Packard E2442A, contact:

Hewlett-Packard Company
2000 South Park Place
Atlanta, GA 30339
(404) 980–7351

Appendix H

Glossary

A

A0–A15: External address pins for data/program memory or I/O devices.

ABU: See a*utobuffering unit (ABU)*.

ACC: See *accumulator (ACC)*.

ACCB: See a*ccumulator buffer (ACCB)*.

ACCH: See *accumulator high byte (ACCH).*

ACCL: See *accumulator low byte (ACCL).*

accumulator (ACC): A 32-bit register that stores the results of an arithmetic logic unit (ALU) operation and provides an input for subsequent ALU operations. The ACC is accessible in two halves: accumulator high (ACCH) and accumulator low (ACCL).

accumulator buffer (ACCB): A 32-bit register that temporarily stores the 32-bit contents of the accumulator (ACC). The ACCB has a direct path back to the arithmetic logic unit (ALU) and can be arithmetically or logically acted upon with the ACC.

accumulator high byte (ACCH): The higher 16 bits stored in the accumulator (ACC). See also *accumulator*.

accumulator low byte (ACCL): The lower 16 bits stored in the accumulator (ACC). See also *accumulator*.

address: The logical location of program code or data stored in memory.

addressing mode: The method by which an instruction calculates the location of its required data.

address visibility (AVIS) bit: A 1-bit field that allows the internal program address to appear at the TMS320C5x pins so that the internal program address can be traced and the interrupt vector can be decoded in conjunction with $\overline{\text{IACK}}$ when the interrupt vectors reside in on-chip memory. At reset, AVIS = 0. This bit is stored in the processor mode status register (PMST).

AFB: See *auxiliary register file bus (AFB)*.

ALU: See *arithmetic logic unit (ALU)*.

analog-to-digital (A/D) converter: An 8-bit successive-approximation converter with internal sample-and-hold circuitry that translates an analog signal to a digital signal.

AR: See *auxiliary register (AR)*.

ARAU: See *auxiliary register arithmetic unit ARAU)*.

ARB: See *auxiliary register buffer (ARB) bits*.

ARCR: See *auxiliary register compare register (ARCR)*.

arithmetic logic unit (ALU): A 32-bit 2s-complement arithmetic logic unit that has two 32-bit input ports and one 32-bit output port feeding the accumulator (ACC). Provides the logic for arithmetic and Boolean operations.

ARP: See *auxiliary register pointer (ARP) bits*.

ARR: See *BSP address receive register (ARR)*.

assembler: A software program that creates a machine-language program from a source file containing assembly language instructions, directives, and macro directives. The assembler substitutes absolute operation codes for symbolic operation codes, and absolute or relocatable addresses for symbolic addresses.

assembly language instructions: The language in which computer operations are represented by mnemonics.

autobuffering receiver enable (BRE) bit: A 1-bit field that enables/disables the autobuffering receiver. At reset, BRE = 0. This bit is stored in the BSP control extension register (SPCE).

autobuffering receiver halt (HALTR) bit: A 1-bit field that enables/disables the autobuffer receiver. At reset, HALTR = 0. This bit is stored in the BSP control extension register (SPCE).

autobuffering transmitter enable (BXE) bit: A 1-bit field that enables/disables the autobuffering transmitter. At reset, BXE = 0. This bit is stored in the BSP control extension register (SPCE).

autobuffering transmitter halt (HALTX) bit: A 1-bit field that enables/disables the autobuffer transmitter. At reset, HALTX = 0. This bit is stored in the BSP control extension extension (SPCE).

autobuffering unit (ABU): An on-chip module that allows the serial port interface to read or write directly to internal memory independently of the central processing unit (CPU). Autobuffering capability can be separately enabled for transmit and receive sections. When autobuffering is disabled, the operation is similar to that of the ’C5x standard serial port.

auxiliary register (AR): Eight 16-bit memory-mapped registers (AR0–AR7) that are used for indirect data address pointers, temporary storage, or integer arithmetic processing through the auxiliary register arithmetic unit (ARAU). Each AR is selected by the auxiliary register pointer (ARP).

auxiliary register arithmetic unit (ARAU): An unsigned 16-bit arithmetic logic unit that calculates indirect addresses using the auxiliary, index, and compare registers as inputs.

auxiliary register buffer (ARB) bits: A 3-bit field that holds the previous value contained in the auxiliary register pointer (ARP). These bits are stored in status register 1 (ST1).

auxiliary register compare register (ARCR): A 16-bit memory-mapped register used as a limit to compare indirect adresses.

auxiliary register file bus (AFB): The bus on which the currently selected auxiliary register (AR) addresses the data memory location.

auxiliary register pointer (ARP) bits: A 3-bit field that selects the auxiliary register (AR) to use in indirect addressing. When the ARP is loaded, the previous ARP value is copied to the auxiliary register buffer (ARB). The ARP can be modified by memory-reference instructions when using indirect addressing, and by the MAR and LST instructions. These bits are stored in status register 0 (ST0).

AVIS: See *address visibility (AVIS) bit*.

AXR: See *BSP address transmit register (AXR)*.

B

barrel shifter: A unit that rotates bits in a word. See also *POSTSCALER* and *PRESCALER*.

BIG bit: A 1-bit field that specifies how the input/out (I/O) port wait-state register is mapped. This bit is stored in the wait-state control register (CWSR). At reset, BIG = 0.

bit-reversed addressing: A method of indirect addressing that allows efficient I/O operations by resequencing the data points in a radix-2 FFT program. The direction of carry propagation in the ARAU is reversed.

BKR: See *BSP receive buffer size register (BKR)*.

BKX: See *BSP transmit buffer size register (BKX)*.

block move address register (BMAR): A 16-bit memory-mapped register that holds an address value for use with block moves or multiply/accumulates.

block repeat active flag (BRAF) bit: A 1-bit field that indicates a block repeat is currently active. This bit is normally set when the RPTB instruction is executed and is cleared when the BRCR register decrements below 0. Writing a 0 to this bit deactivates block repeat. At reset, BRAF = 0. This bit is stored in the processor mode status register (PMST).

block repeat counter register (BRCR): A 16-bit memory-mapped register that limits the number of times a block is repeated.

block repeat program address end register (PAER): A 16-bit memory-mapped register that contains the end address of the segment of code being repeated.

block repeat program address start register (PASR): A 16-bit memory-mapped register that contains the start address of the segment of code being repeated.

BMAR: See *block move address register (BMAR)*.

BOB: See *byte ordering bit (BOB)*.

boot: The process of loading a program into program memory.

boot loader: A built-in segment of code that transfers code from an external source to program memory at power-up.

BRAF: See *block repeat active flag (BRAF) bit*.

BRCR: See *block repeat counter register (BRCR)*.

BRE: See *autobuffering receiver enable (BRE) bit*.

BSP: See *buffered serial port (BSP)*.

BSP address receive register (ARR): An 11-bit memory-mapped register that stores the address for writing a word to be transferred from the data receive register (DRR) to ’C5x internal memory. When autobuffering is enabled (BRE = 1), the ARR is no longer available for software access as a memory-mapped register.

BSP address transmit register (AXR): An 11-bit memory-mapped register that stores the address for reading a word to be transferred from ’C5x internal memory to the data transmit register (DXR). When autobuffering is enabled (BXE = 1), the AXR is no longer available for software access as a memory-mapped register.

BSP control extension register (SPCE): A 16-bit memory-mapped register that contains status and control bits for the buffered serial port (BSP) interface. The 10 LSBs of the SPCE are dedicated to serial port interface control, whereas the 6 MSBs are used for autobuffering unit (ABU) control.

BSP receive buffer size register (BKR): An 11-bit memory-mapped register that stores the address block size for writing a word to be transferred from the data receive register (DRR) to ’C5x internal memory. When autobuffering is enabled (BRE = 1), the BKR is no longer available for software access as a memory-mapped register.

BSP transmit buffer size register (BKX): An 11-bit memory-mapped register that stores the address block size for reading a word to be transferred from ’C5x internal memory to the data transmit register (DXR). When autobuffering is enabled (BXE = 1), the BKX is no longer available for software access as a memory-mapped register.

buffered serial port (BSP): An on-chip module that consists of a full-duplex, double-buffered serial port interface and an autobuffering unit (ABU). The double-buffered serial port of the BSP is an enhanced version of that available in other TMS320C5x devices (’C50, ’C51, ’C52, and ’C53). The double-buffered serial port allows transfer of a continuous communication stream (8-,10-,12- or 16-bit data packets). Status and control of the BSP is specified in the BSP control extension register (SPCE).

burst mode: A synchronous serial port mode in which a single word is transmitted following a frame synchronization pulse (FSX and FSR).

butterfly: A kernel function that computes an N-point fast Fourier transform (FFT), where N is a power of 2. The combinational pattern of inputs resembles butterfly wings.

BXE: See *autobuffering transmitter enable (BXE) bit*.

byte ordering bit (BOB): A 1-bit field that affects host processor data and address transfers when using the host port interface. Only the host processor can toggle this bit. The BOB must be initialized before the first data or address register access. This bit is stored in the HPI control register (HPIC).

C

C: See *carry (C) bit*.

CALU: See *central arithmetic logic unit (CALU)*.

CAR1: See *circular buffer 1 auxiliary register (CAR1) bits*.

CAR2: See *circular buffer 2 auxiliary register (CAR2) bits*.

carry (C) bit: A 1-bit field that stores the carry output of the arithmetic logic unit (ALU). At reset, C = 1. The C bit can be tested by conditional instructions. This bit is stored in status register 1 (ST1).

CBCR: See *circular buffer control register (CBCR)*.

CBER1: See *circular buffer 1 end register (CBER1)*.

CBER2: See *circular buffer 2 end register (CBER2)*.

CBSR1: See *circular buffer 1 start register (CBSR1)*.

CBSR2: See *circular buffer 2 start register (CBSR2)*.

CENB1: See *circular buffer 1 enable (CENB1) bit*.

CENB2: See *circular buffer 2 enable (CENB2) bit*.

central arithmetic logic unit (CALU): A 32-bit arithmetic logic unit that executes 32-bit operations in a single machine cycle. The CALU consists of the arithmetic logic unit (ALU), multiplier (MULT), accumulator (ACC), accumulator buffer (ACCB), and scaling shifters (PRESCALERS, P-SCALER, and POSTSCALER).

central processing unit (CPU): The module of the TMS320C5x that controls and interprets the machine-language program and its execution. The CPU consists of the central arithmetic logic unit (CALU), parallel logic unit (PLU), auxiliary register arithmetic unit (ARAU), and registers.

circular buffer 1 auxiliary register (CAR1) bits: A 3-bit field that identifies which auxiliary register (AR) is assigned to circular buffer 1. These bits are stored in the circular buffer control register (CBCR).

circular buffer 1 enable (CENB1) bit: A 1-bit field that enables/disables circular buffer 1. At reset, CENB1 = 0. This bit is stored in the circular buffer control register (CBCR).

circular buffer 1 end register (CBER1): A 16-bit memory-mapped register that indicates the circular buffer 1 end address.

circular buffer 1 start register (CBSR1): A 16-bit memory-mapped register that indicates the circular buffer 1 start address.

circular buffer 2 auxiliary register (CAR2) bits: A 3-bit field that identifies which auxiliary register (AR) is assigned to circular buffer 2. These bits are stored in the circular buffer control register (CBCR).

circular buffer 2 enable (CENB2) bit: A 1-bit field that enables/disables circular buffer 2. At reset, CENB2 = 0. This bit is stored in the circular buffer control register (CBCR).

circular buffer 2 end register (CBER2): A 16-bit memory-mapped register that indicates the circular buffer 2 end address.

circular buffer 2 start register (CBSR2): A 16-bit memory-mapped register that indicates the circular buffer 2 start address.

circular buffer control register (CBCR): An 8-bit memory-mapped register that enables/disables the circular buffers (CENB1 and CENB2 bits) and defines which auxiliary registers (CAR1 and CAR2 bits) are mapped to the circular buffers.

CLKDV: See *internal transmit clock division factor (CLKDV) bits*.

CLKP: See *clock polarity (CLKP) bit*.

clock modes: Options used by the clock generator to change the internal CPU clock frequency to a fraction or multiple of the frequency of the input clock signal.

clock mode (MCM) bit: A 1-bit field that specifies the source of the clock for CLKX. At reset, MCM = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

clock polarity (CLKP) bit: A 1-bit field that indicates when the data is sampled by the receiver and sent by the transmitter. At reset, CLKP = 0. This bit is stored in the BSP control extension register (SPCE).

CNF: See *configuration control (CNF) bit*.

code: A set of instructions written to perform a task.

cold boot: The process of loading a program into program memory at power-up.

configuration control (CNF) bit: A 1-bit field that indicates if on-chip dual-access RAM block 0 (DARAM B0) is mapped to program or data space. At reset, CNF = 0. This bit is stored in status register 1 (ST1).

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.

continuous mode: A synchronous serial port mode in which only one frame synchronization pulse (FSX and FSR) is necessary to transmit several packets at maximum frequency.

CPU: See *central processing unit (CPU)*.

current auxiliary register: The auxiliary register pointed to by the auxiliary register pointer (ARP).

CWSR: See *wait-state control register (CWSR)*.

D

D0–D15: External data bus pins that transfer data between the 'C5x and external data/program memory or I/O devices.

DAB: See *direct address bus (DAB)*.

DARAM: See *dual-access RAM*.

data bus: A group of connections used to route data.

data memory: A memory region used for storing and manipulating data.

data memory address (dma): The seven LSBs of a direct addressed instruction that contains the immediate relative address within a 128-word data page. The seven LSBs are concatenated with the data memory page pointer (DP) to form the direct memory address of 16 bits. See also *data memory page pointer (DP)*.

data memory page pointer (DP) bits: A 9-bit field that specifies the current data memory page address. The DP bits are concatenated with the 7 LSBs of the instruction word to form the direct memory address of 16 bits. These bits are stored in status register 0 (ST0).

data memory page 0: The first page in data memory space where the memory-mapped registers and the scratch-pad RAM block (B2) reside.

data receive register (DRR): A 16-bit memory-mapped register that holds serial data copied from the receive shift register (RSR). When autobuffering is enabled (BRE = 1), the DRR is no longer available for software access as a memory-mapped register. See also *data receive shift register (RSR)*.

data receive shift register (RSR): A 16-bit register that holds serial data received from the DR pin. See also *data receive register (DRR)*.

data transmit register (DXR): A 16-bit memory-mapped register that holds serial data to be copied to the data transmit shift register (XSR). When autobuffering is enabled (BXE = 1), the DXR is no longer available for software access as a memory-mapped register. See also *data transmit shift register (XSR)*.

data transmit shift register (XSR): A 16-bit register that holds serial data to be transmitted from the DX pin (or TDX pin when TDM = 1). See also *data transmit register (DXR)* and *TDM data transmit register (TDXR)*.

DBMR: See *dynamic bit manipulation register (DBMR)*.

digital loopback (DLB) mode: A synchronous serial port test mode in which the DLB bit connects the receive pins to the transmit pins on the same device to test if the port is operating correctly.

digital loopback mode (DLB) bit: A 1-bit field that puts the serial port in digital loopback mode. At reset, DLB = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

direct address bus (DAB): A 16-bit bus that provides the data address used by the central processing unit (CPU).

direct memory access (DMA): A mode where a device other than the host processor contends for, and receives, mastership of the memory bus so that data transfers may take place independent of the host.

DLB: See *digital loopback mode (DLB) bit*.

dma: See *data memory address (dma)*.

DMA: See *direct memory access (DMA)*.

DP: See *data memory page pointer (DP) bits*.

DRR: See *data receive register (DRR)*.

DSP interrupt (DSPINT) bit: A 1-bit field that enables/disables an interrupt from a host processor to the TMS320C57. The DSPINT bit is written from the host processor; a ’C57 write has no effect on the DSPINT bit. When DSPINT = 1, a ’C57 interrupt is generated. The host must write a 0 to the DSPINT bit while writing to the BOB or HINT bits, so that the host does not provoke an unwanted ’C57 interrupt. This bit is stored in the HPI control register (HPIC).

DSPINT: See *DSP interrupt (DSPINT) bit*.

dual-access RAM (DARAM): Memory space that can be read from and written to in the same clock cycle.

dynamic bit manipulation register (DBMR): A 16-bit memory-mapped register that masks the input to the parallel logic unit (PLU) in the absence of a long immediate value.

DXR: See *data transmit register (DXR)*.

E

enable extra index register (NDX) bit: A 1-bit field that determines if a modification or write to auxiliary register 0 (AR0) also modifies or writes to the index register (INDX), and the auxiliary register compare register (ARCR) to maintain compatibility with the TMS320C2x. This bit is stored in the processor mode status register (PMST).

enable multiple TREGs (TRM) bit: A 1-bit field that indicates if an LT(A,D,P,S) instruction loads only TREG0 or loads all three of the temporary registers (TREG0, TREG1, and TREG2) to maintain compatibility with the TMS320C2x. The TRM bit allows the TMS320C5x to operate in either 'C2x-compatible mode (TRM = 0) or 'C5x-enhanced mode (TRM = 1) in conjunction with the use of TREG0, TREG1, and TREG2. The TRM bit affects the operation of all 'C2x-compatible instructions that modify TREG0. This bit is stored in the processor mode status register (PMST).

external flag (XF) pin status bit: A 1-bit field that drives the level of the external flag (XF) pin. At reset, XF = 1.This bit is stored in status register 1 (ST1).

external interrupt: A hardware interrupt triggered by a pin ($\overline{\text{INT1}}$–$\overline{\text{INT4}}$).

F

fast Fourier transform (FFT): An efficient method of computing the discrete Fourier transform, which transforms functions between the time domain and frequency domain. The time-to-frequency domain is called the forward transform, and the frequency-to-time domain is called the inverse transformation. See also *butterfly*.

FE: See *format extension (FE) bit*.

FFT: See *fast Fourier transform (FFT)*.

FIG: See *frame ignore (FIG) bit*.

FO: See *format (FO) bit*.

format (FO) bit: A 1-bit field that specifies the word length of the serial port transmitter and receiver. The data is transferred with the MSB first. At reset, FO = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

format extension (FE) bit: A 1-bit field used in conjunction with the format bit (FO) to specify the word length of the BSP serial port transmitter and receiver. When FO = FE = 00, the format is 16-bit words; when FO = FE = 01, the format is 10-bit words; when FO = FE = 10, the format is 8-bit words; and when FO = FE = 11, the format is 12-bit words. For 8-,10-, and 12-bit words, the received words are right-justified and the sign bit is extended to form a 16-bit word. The words to transmit must be right-justified. At reset, FE = 0. This bit is stored in the BSP control extension register (SPCE).

frame ignore (FIG) bit: A 1-bit field used only in transmit continuous mode with external frame and in receive continuous mode. At reset, FIG = 0. This bit is stored in the BSP control extension register (SPCE).

frame synchronization mode (FSM) bit: A 1-bit field that specifies whether frame synchronization pulses (FSX and FSR) are required for serial port operation. At reset, FSM = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

frame synchronization polarity (FSP) bit: A 1-bit field that determines the status of the frame synchronization pulses. At reset, FSP = 0. This bit is stored in the BSP control extension register (SPCE).

Free bit: A 1-bit field used in conjunction with the Soft bit to determine the state of the serial port clock when a breakpoint is encountered in the high-level language debugger. At reset, Free = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

FSM: See *frame synchronization mode (FSM) bit*.

FSP: See *frame synchronization polarity (FSP) bit*.

G

general-purpose input/output pins: Pins that can be used to supply input signals from an external device or output signals to an external device. These pins are not linked to specific uses; rather, they provide input or output signals for a variety of purposes. These pins include the general-purpose $\overline{\text{BIO}}$ input pin and XF output pin.

global data memory space: One of four memory spaces. The global data memory space can either share data with other processors within the system or serve as additional data memory space.

global memory allocation register (GREG): An 8-bit memory-mapped register that specifies the size of the global memory space. At reset, the GREG is cleared.

GREG: See *global memory allocation register (GREG)*.

H

HALTR: See *autobuffering receiver halt (HALTR) bit*.

HALTX: See *autobuffering transmitter halt (HALTX) bit*.

hardware interrupt: An interrupt triggered through physical connections with on-chip peripherals or external devices.

HINT bit: *'C57-to-Host Processor Interrupt*. A 1-bit field that enables/disables an interrupt from the TMS320C57 to a host processor. At reset, HINT = 0. This bit is stored in the HPI control register (HPIC).

HM: See *hold mode (HM) bit*.

HOM: See *host-only mode (HOM)*.

hold mode (HM) bit: A 1-bit field that determines whether the central processing unit (CPU) can stop or continue when the $\overline{\text{HOLD}}$ signal initiates a power-down mode. At reset, HM = 1. This bit is stored in status register 1 (ST1).

host-only mode (HOM): The mode that allows the host to access HPI memory while the TMS230C57 is in IDLE2 (all internal clocks stopped) or in reset mode. The external 'C57 clock may even be stopped. The host can therefore access the HPI RAM while the 'C57 is in its optimum configuration in terms of power consumption.

host port interface (HPI): An on-chip module consisting of an 8-bit parallel port that interfaces a host processor to the TMS320C57. The HPI has two modes of operation, shared-access mode (SAM) and host-only mode (HOM). Status and control of the HPI is specified in the HPI control register (HPIC). See also *shared-access mode (SAM)* and *host-only mode (HOM)*.

HPI: See *host port interface (HPI)*.

HPIA: See *HPI address register (HPIA)*.

HPIAH: See *HPI address register high byte (HPIAH)*.

HPIAL: See *HPI address register low byte (HPIAL)*.

HPIC: See *HPI control register (HPIC)*.

HPICH: See *HPI control register high byte (HPICH)*.

HPICL: See *HPI control register low byte (HPICL)*.

HPI address register (HPIA): A 16-bit register that stores the address of the host port interface (HPI) memory block. The HPIA can be preincremented or postincremented.

HPI address register high byte (HPIAH): The higher 16 bits stored in the HPI address register (HPIA). See also *HPI address register (HPIA)*.

HPI address register low byte (HPIAL): The lower 16 bits stored in the HPI address register (HPIA). See also *HPI address register (HPIA)*.

HPI control register (HPIC): A 16-bit register that contains status and control bits for the host port interface (HPI).

HPI control register high byte (HPICH): The higher 16 bits stored in the HPI control register (HPIC). See also *HPI control register (HPIC)*.

HPI control register low byte (HPICL): The lower 16 bits stored in the HPI control register (HPIC). See also *HPI control register (HPIC)*.

I

I/O port wait-state register (IOWSR): A 16-bit memory-mapped register that specifies the number of wait states for the input/out (I/O) port. The IOWSR can be mapped in one of two ways as specified by the BIG bit in the wait-state control register (CWSR). At reset, IOWSR = FFFF.

IFR: See *interrupt flag register (IFR)*.

IMR: See *interrupt mask register (IMR)*.

IN0 bit: *Input 0 bit.* A 1-bit field that allows the CLKR pin to be used as an input. IN0 reflects the current level of the CLKR pin of the device. This bit is stored in the SPC and TDM serial port control register (TSPC).

IN1 bit: *Input 1 bit.* A 1-bit field that allows the CLKX pin to be used as an input. IN1 reflects the current level of the CLKX pin of the device. This bit is stored in the SPC and TDM serial port control register (TSPC).

index register (INDX): A 16-bit memory-mapped register that specifies increment sizes greater than 1 for indirect addressing updates. In bit-reversed addressing, the INDX defines the array size.

INDX: See *index register (INDX)*.

instruction: The basic unit of programming that causes the execution of one operation; it consists of an opcode and operands along with optional labels and comments.

instruction register (IREG): A 16-bit register that contains the actual instruction being executed.

internal interrupt: A hardware interrupt caused by an on-chip peripheral.

internal transmit clock division factor (CLKDV) bits: A 5-bit field that determines the internal transmit clock duty cycle. At reset, CLKDV = 00011. These bits are stored in the BSP control extension register (SPCE).

interrupt: An exceptional condition that is caused either by an external event to the CPU or by a previously executed instruction that forces the current program to stop. The CPU executes instructions of an interrupt service routine (ISR) at an address corresponding to the source of the interrupt. After the CPU services the interrupt, the CPU resumes execution of the program at the instruction whose execution was interrupted.

interrupt flag register (IFR): A 16-bit memory-mapped register that flags pending interrupts. The IFR may be read to identify pending interrupts and written to clear selected interrupts. A 1 read from any IFR bit position indicates a pending interrupt. A 1 written to any IFR bit position clears the corresponding interrupt. A 0 written to any IFR bit position has no effect. At reset, the IFR is cleared.

interrupt mask register (IMR): A 16-bit memory-mapped register that masks external and internal interrupts. The IMR may be read and written to. A 1 written to any IMR bit position enables the corresponding interrupt (when INTM = 0).

interrupt mode (INTM) bit: A 1-bit field that globally masks or enables all interrupts. When INTM = 0, all unmasked interrupts are enabled. When INTM = 1, all maskable interrupts are disabled. INTM has no effect on the nonmaskable $\overline{RS}$ and $\overline{NMI}$ interrupts. At reset, INTM = 1.This bit is stored in status register 0 (ST0).

interrupt service routine (ISR): A module of code that is executed in response to a hardware or software interrupt.

interrupt vector pointer (IPTR) bits: A 5-bit field that identifies the 2K page where the interrupt vectors currently reside in the system. The IPTR lets you remap the interrupt vectors to RAM for boot-loaded operations. At reset, IPTR = 0. These bits are stored in the processor mode status register (PMST).

INTM: See *interrupt mode (INTM) bit*.

IOWSR: See *I/O Port Wait-State Register (IOWSR)*.

IPTR: See *interrupt vector pointer (IPTR) bits*.

IREG: See *instruction register (IREG)*.

ISR: See *interrupt service routine (ISR)*.

L

latency: The delay between when a condition occurs and when the device reacts to the condition. Also, in a pipeline, the delay between the execution of two instructions that is necessary to ensure that the values used by the second instruction are correct.

LSB: *least significant bit*. The lowest-order bit in a word.

M

maskable interrupts: A hardware interrupt that can be enabled or disabled through software.

MCM: See *clock mode (MCM) bit*.

MCS: See *microcall stack (MCS)*.

memory map: A map of the addressable memory space accessed by the TMS320C5x processor partitioned according to functionality (memory, registers, etc.).

memory-mapped registers: The TMS320C5x processor has 96 registers mapped into page 0 of the data memory space. There are 28 core CPU registers, 17 peripheral registers, 16 input/output (I/O) port registers, and 35 reserved registers.

microcall stack (MCS): A single-word stack that temporarily stores the contents of the prefetch counter (PFC) while the PFC addresses data memory with the block move (BLDD/BLPD), multiply-accumulate (MAC/MACD), and table read/write (TBLR/TBLW) instructions.

microprocessor/microcomputer (MP/$\overline{\text{MC}}$) bit: A 1-bit field that indicates if on-chip ROM is mapped into program address space. When MP/$\overline{\text{MC}}$ = 0, the on-chip ROM is enabled. When MP/$\overline{\text{MC}}$ = 1, the on-chip ROM is not addressable. At reset, the MP/$\overline{\text{MC}}$ bit is set to the value corresponding to the logic level on the MP/$\overline{\text{MC}}$ pin. The level on the MP/$\overline{\text{MC}}$ pin is sampled at reset only and has no effect until the next reset. This bit is stored in the processor mode status register (PMST).

mnemonic: An alphanumeric symbol designed to aid human memory; it commonly represents the operation code of an assembly language instruction name that the assembler translates into machine code.

MP/$\overline{\text{MC}}$: See *microprocessor/microcomputer (MP/$\overline{\text{MC}}$) bit*.

MSB: *most significant bit*. The highest-order bit in a word.

MULT: See *multiplier (MULT)*.

multiplier (MULT): A 16-by-16-bit multiplier that generates a 32-bit product. The multiplier executes multiple operations in a single machine cycle and operates using either signed or unsigned 2s-complement arithmetic. The operand for the multiplier is specified by the value in temporary register 0 (TREG0). The result of the multiplier is stored in the product register (PREG).

N

nested interrupt: A higher-priority interrupt that must be serviced before completion of the current interrupt service routine (ISR). An executing ISR can set the interrupt mask register (IMR) bits to prevent being suspended by another interrupt.

NDX: See *enable extra index register (NDX) bit*.

nonmaskable interrupt: An interrupt that can be neither masked by the interrupt mask register (IMR) nor disabled by the INTM bit of status register ST0.

O

off-chip: A device external to the TMS320C5x device.

on-chip: An element or module of the TMS320C5x device.

opcode: *operation code.* In most cases, the first byte of the machine code that describes the type of operation and combination of operands to the central processing unit (CPU).

operand: The part of an instruction that designates where the central processing unit (CPU) will fetch or store data. The operand consists of the arguments, or parameters, of an assembly language instruction, assembler directive, or macro directive.

OV: See *overflow (OV) bit*.

overflow: A condition in which the result of an arithmetic operation exceeds the capacity of the register used to hold that result.

overflow (OV) bit: A 1-bit flag that indicates an arithmetic operation overflow in the arithmetic logic unit (ALU). At reset, OV = 0.This bit is stored in status register 0 (ST0).

overflow mode (OVM) bit: A 1-bit field that determines if an overflow in the arithmetic logic unit (ALU) will wrap around or saturate. This bit is stored in status register 0 (ST0).

OVLY: See *RAM overlay (OVLY) bit*.

OVM: See *overflow mode (OVM) bit*.

P

PAER: See *block repeat program address end register (PAER)*.

parallel logic unit (PLU): A 16-bit logic unit that executes logic operations from either long immediate operands or the contents of the dynamic bit manipulation register (DBMR) directly upon data locations without affecting the contents of the accumulator (ACC) or product register (PREG).

PASR: See *block repeat program address start register (PASR)*.

PC: See *program counter (PC)*.

PCM: See *pulse coded modulation mode (PCM) bit*.

PDWSR: See *program/data wait-state register (PDWSR)*.

PFC: See *prefetch counter (PFC)*.

pipelining: A design technique for reducing the effective propagation delay per instruction operation by partitioning the operation into a series of four independent stages, each of which performs a portion of the operation.

PLU: See *parallel logic unit (PLU)*.

PM: See *product shift mode (PM) bits*.

PMST: See *processor mode status register (PMST)*.

pop: Action of removing a word from a stack.

POSTSCALER: *postscaling shifter*. A 0- to 7-bit left barrel shifter used to postscale data coming out of the accumulator (ACC).

PRD: See *timer period register (PRD)*.

prefetch counter (PFC): A 16-bit register that prefetches program instructions. The PFC contains the address of the instruction currently being prefetched and is updated when a new prefetch is initiated.

PREG: See *product register (PREG)*.

PRESCALER: *prescaling shifter*. A 0- to 16-bit left barrel shifter used to prescale data coming into the arithmetic logic unit (ALU). This shifter is also used as a 0- to 16-bit right barrel shifter of the accumulator (ACC). The shift count is specified by a constant in the instruction or by the value in temporary register 1 (TREG1).

processor mode status register (PMST): A 16-bit memory-mapped register that contains status and control bits.

product register (PREG): A 32-bit register that holds the output from the multiplier. The high and low words of the PREG can be accessed individually. See also *multiplier (MULT)*.

product shift mode (PM) bits: A 2-bit field that defines the product shifter (P-SCALER) mode. These two bits determine the shift value (0-, 1-, 4-bit left shifter, 6-bit right shifter) for the output of the product register (PREG).These bits are stored in status register 1 (ST1).

program/data wait-state register (PDWSR): (For the TMS320C50, ’C51, and ’C53) a 16-bit memory-mapped register that specifies the number of wait states for the program and data space. The higher byte of PDWSR specifies the data space wait states and the lower byte specifies the program space wait states. At reset, PDWSR = FFFF.

(For the TMS320C52, ’C56, and ’C57) a 16-bit memory-mapped register that specifies the number of wait states for the program, data, and input/output (I/O) space. Bits 0–2 of PDWSR specify the program space wait states, bits 3–5 specify the data space wait states, bits 6–8 specify the I/O space wait states, and bits 9–15 are reserved. At reset, PDWSR = FFFF.

program controller: Logic circuitry that decodes instructions, manages the pipeline, stores the central processing unit (CPU) status, and decodes conditional operations.

program counter (PC): A 16-bit register that identifies the current statement in the program. The PC addresses program memory sequentially and always contains the address of the next instruction to be fetched. The PC contents are updated following each instruction decode operation.

P-SCALER: *Product Shifter.* A 0-, 1-, or 4-bit left shifter that removes extra signed bits (gained in the multiply operation) when fixed-point arithmetic is used; or a 6-bit right shifter that scales the products down to avoid overflow in the accumulation process. The shift mode is specified by the product shift mode (PM) bits.

PSC: See *timer prescaler counter (PSC) bits.*

pulse coded modulation mode (PCM) bit: A 1-bit field that enables/disables the BSP transmitter. This bit is stored in the BSP control extension register (SPCE).

push: Action of placing a word onto a stack.

R

RAM overlay (OVLY) bit: A 1-bit field that determines if on-chip single-access RAM is addressable in data memory space. At reset, OVLY = 0. This bit is stored in the processor mode status register (PMST).

receive buffer half received (RH) bit: A 1-bit flag that indicates which half of the receive buffer has been received. At reset, RH = 0. This bit is stored in the BSP control extension register (SPCE).

receive ready (RRDY) bit: A 1-bit flag that transitions from 0 to 1 to indicate the data receive shift register (RSR) contents have been copied to the data receive register (DRR) and that data can be read. A receive interrupt (RINT) is generated upon the transition. The RRDY bit can be polled in software in lieu of using serial port interrupts. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

receiver reset ($\overline{\text{RRST}}$) bit: A 1-bit flag that resets the serial port receiver. At reset, $\overline{\text{RRST}}$ = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

receive shift register full (RSRFULL) bit: A 1-bit flag that indicates if the serial port receiver has experienced overrun. This bit is stored in the serial port control register (SPC).

register: A group of bits used for temporarily holding data or for controlling or specifying the status of a device.

repeat counter register (RPTC): A 16-bit memory-mapped register that controls the repeated execution of a single instruction.

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 start at a specified address.

RH: See *receive buffer half received (RH) bit*.

RINT: See *serial port receive interrupt (RINT) bit*.

RPTC: See *repeat counter register (RPTC)*.

RRDY: See *receive ready (RRDY) bit*.

$\overline{\text{RRST}}$: See *receiver reset ($\overline{\text{RRST}}$) bit*.

RSR: See *data receive shift register (RSR)*.

RSRFULL: See *receive shift register full (RSRFULL) bit*.

S

SAM: See *shared-access mode (SAM)*.

SARAM: See *single-access RAM (SARAM)*.

scratch-pad RAM: Block 2 (B2) on data memory page 0 in local data space (32 words) of DARAM. Scratch-pad RAM supports dual-access operations and can be addressed via any data memory addressing mode.

serial port control register (SPC): A 16-bit memory-mapped register that contains status and control bits for the serial port interface. The SPC is identical to the time-division multiplexed serial port control register (TSPC), except that bit 0 is reserved for the TDM bit.

serial port interface: An on-chip full-duplex serial port interface that provides direct serial communication to serial devices with a minimum of external hardware, such as codecs and serial analog-to-digital (A/D) converters. Status and control of the serial port is specified in the serial port control register (SPC).

serial port receive interrupt (RINT) bit: A 1-bit flag that indicates the data receive shift register (RSR) contents have been copied to the data receive register (DRR). This bit is stored in the interrupt flag register (IFR).

serial port transmit interrupt (XINT) bit: A 1-bit flag that indicates the the data transmit register (DXR) contents has been copied to the data transmit shift register (XSR). This bit is stored in the interrupt flag register (IFR).

shared-access mode (SAM): The mode that allows both the TMS320C57 and the host to access HPI memory. In this mode, asynchronous host accesses are synchronized internally and, in case of conflict, the host has access priority and the 'C57 waits one cycle.

shared-access mode (SMOD) bit: A 1-bit field that enables/disables the shared access mode (SAM). This bit is stored in the HPI control register (HPIC). See also *shared-access mode (SAM)* and *host-only mode (HOM)*.

shifter: A unit that shifts bits in a word to the left or to the right. See also *P-SCALER*.

sign-extension: The process of filling the high-order bits of a number with the sign bit, when loading a 16-bit number into a 32-bit field.

sign-extension mode (SXM) bit: A 1-bit field that enables/disables sign extension of an arithmetic operation. This bit is stored in status register 1 (ST1).

single-access RAM (SARAM): Memory space that only can be read from or written to in a single clock cycle.

SMOD: See *shared-access mode (SMOD) bit*.

Soft bit: A 1-bit field used in conjunction with the Free bit to determine the state of the serial port clock when a breakpoint is encountered in the high-level language debugger. At reset, Soft = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

software interrupt: An interrupt caused by the execution of an INTR, NMI, or TRAP instruction.

SPC: See *serial port control register (SPC)*.

SPCE: See *BSP control extension register (SPCE)*.

stack: An 8-level-deep by 16-bit hardware stack used as a last-in, first-out memory for temporary variable storage; used during interrupt service routines (ISR) and calls to store the current program status. The area occupied by the stack is determined by the stack pointer and the application program.

status register: A 16-bit register that contains status and control bits.

SXM: See *sign-extension mode (SXM) bit*.

T

TADD: See *TDM address (TADD)*.

TC: See *test/control (TC) bit*.

TCLK: See *TDM clock (TCLK)*.

TCR: See *timer control register (TCR)*.

TCSR: See *TDM channel select register (TCSR)*.

TDAT: See *TDM data (TDAT)*.

TDDR: See *timer divide-down register (TDDR) bits*.

TDM: See *time-division multiplexed (TDM) bit*.

TDM address (TADD): A single, bi-directional address line that identifies which devices on the four-wire serial bus should read in the data on the TDM data (TDAT) line.

TDM channel select register (TCSR): A 16-bit memory-mapped register that specifies in which of the eight time slots (channels) a device on the four-wire serial bus is to transmit. A 1 in any one or more of bits 0–7 of the TCSR sets the device transmitter active during the corresponding time slot. Bits 8–15 are reserved.

TDM clock (TCLK): A single, bi-directional clock line for TDM operation. The TDM receive clock (TCLKR) and TDM transmit clock (TCLKX) pins are externally connected to form the TCLK line.

TDM data (TDAT): A single, bi-directional line from which all TDM data is carried. The TDM serial data receive (TDR) and TDM serial data transmit (TDX) pins are externally connected to form the TDAT line.

TDM data receive register (TRCV): A 16-bit memory-mapped register that holds serial data copied from the TDM receive shift register (TRSR). When multiprocessing is enabled (TDM = 1), the TRCV is no longer available for software access as a memory-mapped register. See also *TDM data receive shift register (TRSR)*.

TDM data receive shift register (TRSR): A 16-bit register that holds serial data received from the TDM data (TDAT) line. See also *TDM data receive register (TRCV)*.

TDM data transmit register (TDXR): A 16-bit memory-mapped register that holds serial data to be copied to the data transmit shift register (XSR). When multiprocessing is enabled (TDM = 1), the TDXR is no longer available for software access as a memory-mapped register. See also *data transmit shift register (XSR)*.

TDM receive address register (TRAD): A 16-bit memory-mapped register that contains various information regarding the status of the TDM address (TADD) line and verifies the relationship between instruction cycles and TDM port timing.

TDM receive interrupt (TRNT) bit: A 1-bit flag that indicates the TDM data receive shift register (TRSR) contents have been copied to the TDM data receive register (TRCV). This bit is stored in the interrupt flag register (IFR).

TDM receive/transmit address register (TRTA): A 16-bit memory-mapped register that specifies to which device(s) on the four-wire serial bus a given device can transmit. The lower byte of the TRTA specifies the receive address (RA) of the device and the higher byte specifies the transmit address (TA). The address is sent over the TDM address (TADD) line.

TDM serial port control register (TSPC): A 16-bit memory-mapped register that contains status and control bits for the TDM serial port interface. The TSPC is identical to the serial port interface control register (SPC), except for the TDM bit 0.

TDM transmit interrupt (TXNT) bit: A 1-bit flag that indicates the TDM data transmit register (TDXR) contents have been copied to the data transmit shift register (XSR). This bit is stored in the interrupt flag register (IFR).

TDXR: See *TDM data transmit register (TDXR)*.

temporary register: A 16-bit register that holds a temporary data value. See also *TREG0*, *TREG1*, and *TREG2*.

test/control (TC) bit: A 1-bit flag that stores the results of the arithmetic logic unit (ALU) or parallel logic unit (PLU) test bit operations. The TC bit is affected by the APL, BIT, BITT, CMPR, CPL, LST #1, NORM, OPL, and XPL instructions. The status of the TC bit influences the execution of the conditional branch, call, and return instructions. This bit is stored in status register 1 (ST1).

TIM: See *timer counter register (TIM)*.

time-division multiplexed (TDM) bit: A 1-bit field that enables/disables the TDM serial port. This bit is stored in the TDM serial port control register (TSPC).

time-division multiplexing (TDM): The process by which a single serial bus is shared by up to eight TMS320C5x devices with each device taking turns to communicate on the bus. There are a total of eight time slots (channels) available. During a time slot, a given device may talk to any combination of devices on the bus.

timer control register (TCR): A 16-bit memory-mapped register that contains status and control bits for the on-chip timer.

timer counter register (TIM): A 16-bit memory-mapped register that specifies the current count for the on-chip timer. The TIM is decremented once after each timer prescaler counter (PSC) decrement past 0. When the TIM is decremented past 0 or the timer is reset, the TIM is loaded with the contents of the timer period register (PRD) and an internal timer interrupt (TINT) is generated.

timer divide-down register (TDDR) bits: A 4-bit field that specifies the timer divide-down ratio (period) for the on-chip timer. When the timer prescaler counter (PSC) is decremented past 0, the PSC is loaded with the contents of the TDDR. At reset, TDDR = 0000. These bits are stored in the timer control register (TCR).

timer interrupt (TINT) bit: A 1-bit flag that indicates the timer counter register (TIM) has decremented past 0. This bit is stored in the interrupt flag register (IFR).

timer period register (PRD): A 16-bit memory-mapped register that specifies the period for the on-chip timer. When the timer counter register (TIM) is decremented past 0, the TIM is loaded with the contents of the PRD.

timer prescaler counter (PSC) bits: A 4-bit field that specifies the count for the on-chip timer. When the PSC is decremented past 0 or the timer is reset, the PSC is loaded with the contents of the timer divide-down register (TDDR) and the timer counter register (TIM) is decremented. These bits are stored in the timer control register (TCR).

timer reload (TRB) bit: A 1-bit flag that resets the on-chip timer. When TRB = 1, the timer counter register (TIM) is loaded with the value in the timer period register (PRD) and the timer prescaler counter (PSC) is loaded with the value of the timer divide-down register (TDDR) bits. This bit is stored in the timer control register (TCR).

timer stop status (TSS) bit: A 1-bit flag that stops and restarts the on-chip timer. At reset, TSS = 0 and the timer immediately starts timing. This bit is stored in the timer control register (TCR).

TINT: See *timer interrupt (TINT) bit*.

TRAD: See *TDM receive address register (TRAD)*.

transmit buffer half transmitted (XH) bit: A 1-bit flag that indicates which half of transmit buffer transmitted. The XH bit can be read when an XINT interrupt occurs (interrupt program or IFR polling). At reset, XH = 0. This bit is stored in the BSP control extension register (SPCE).

transmit mode bit (TXM) bit: A 1-bit field that specifies the source of the frame synchronization transmit (FSX) pulse. At reset, TXM = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

transmit ready (XRDY) bit: A 1-bit flag that transitions from 0 to 1 to indicate the data transmit register (DXR) contents have been copied to the data transmit shift register (XSR) and that data is ready to be loaded with a new data word. A transmit interrupt (XINT) is generated upon the transition. The XRDY bit can be polled in software in lieu of using serial port interrupts. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

transmit shift register empty ($\overline{\text{XSREMPTY}}$) bit: A 1-bit flag that indicates if the serial port transmitter has experienced underflow. This bit is stored in the serial port control register (SPC).

transmitter reset ($\overline{\text{XRST}}$) bit: A 1-bit flag that resets the serial port transmitter. At reset, $\overline{\text{XRST}}$ = 0. This bit is stored in the serial port control register (SPC) and TDM serial port control register (TSPC).

TRB: See *timer reload (TRB) bit*.

TRCV: See *TDM data receive register (TRCV)*.

TREG0: *temporary register 0*. A 16-bit memory-mapped register that holds an operand for the multiplier. See also *multiplier (MULT)*.

TREG1: *temporary register 1*. A 5-bit memory-mapped register that holds a dynamic prescaling shift count for data inputs to the arithmetic logic unit (ALU). See also *PRESCALER*.

TREG2: *temporary register 2*. A 4-bit memory-mapped register that holds a dynamic bit pointer for the BITT instruction.

TRM: See *enable multiple TREGs (TRM) bit*.

TRNT: See *TDM receive interrupt (TRNT) bit*.

TRSR: See *TDM data receive shift register (TRSR)*.

TRTA: See *TDM receive/transmit address register (TRTA)*.

TSPC: See *TDM serial port control register (TSPC)*.

TSS: See *timer stop status (TSS) bit*.

TXM: See *transmit mode (TXM) bit*.

TXNT: See *TDM transmit interrupt (TXNT) bit*.

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 (one CLKOUT1 cycle) for every wait state.

wait-state control register (CWSR): A 5-bit memory-mapped register that controls the mapping of the program/data wait-state register (PDWSR), the input/output port wait-state register (IOWSR), and the number of wait states. At reset, CWSR = 01111_2.

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). Wait states are set in the wait-state control register (CWSR).

warm boot: The process by which the processor transfers control to the entry address of a previously-loaded program.

word: A word, as defined in this document, consists of a sequence of 16 adjacent bits (two bytes).

X

XF: See *external flag (XF) pin status bit*.

XH: See *transmit buffer half transmitted (XH) bit*.

XINT: See *serial port transmit interrupt (XINT) bit*.

XRDY: See *transmit ready (XRDY) bit*.

$\overline{\text{XRST}}$: See *transmitter reset ($\overline{\text{XRST}}$) bit*.

XSR: See *data transmit shift register (XSR)*.

$\overline{\text{XSREMPTY}}$: See *transmit shift register empty ($\overline{\text{XSREMPTY}}$) bit*.

Z

zero fill: A method of filling the low- or high-order bits with zeros when a shift occurs.

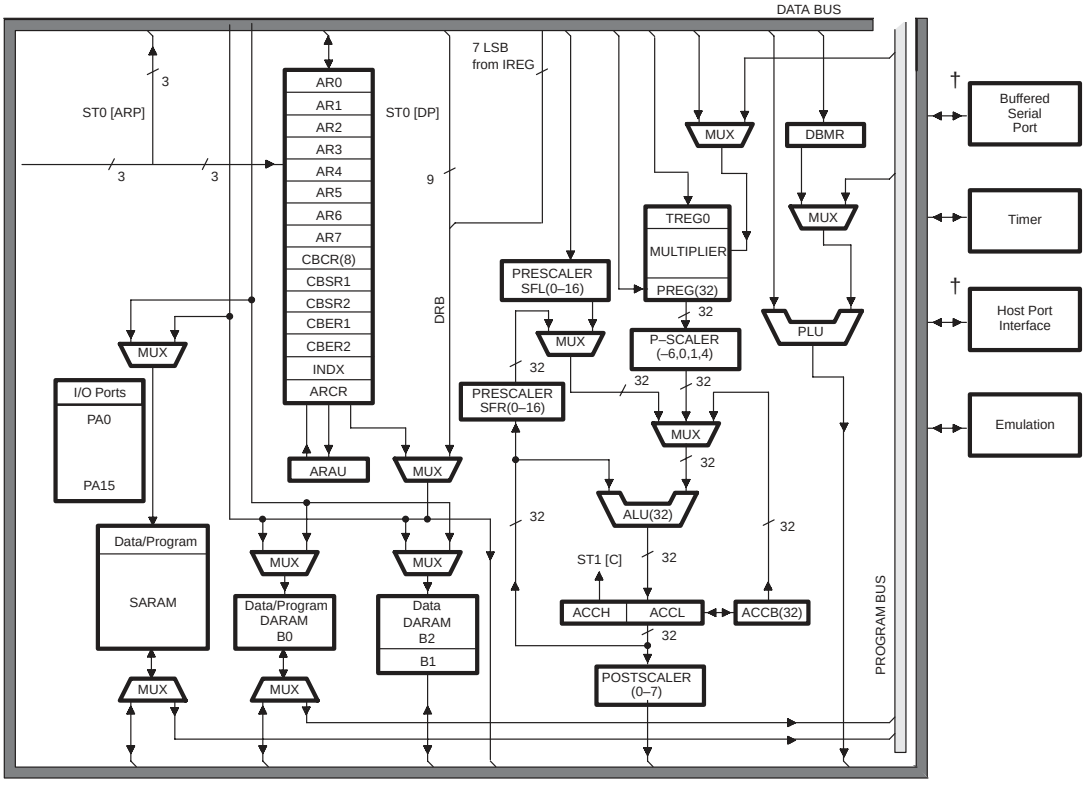
Appendix I

Summary of Updates in This Document

This appendix provides a summary of the updates in this version of the document. Updates within paragraphs appear in a **bold typeface**.

Page:	Change or Add:
3–3	In the bottom half of Figure 3–1, the auxiliary register file MUX output now connects with the trailing wire bus found on the data bus.

Figure 3–1. Block Diagram of ’C5x DSP – Central Processing Unit (CPU)



Notes: All registers and data lines are 16-bits wide unless otherwise specified.
† Not available on all devices.

Page: **Change or Add:**

4–11 In Table 4–5, changed the reset values for the ARP bit and the OVM bit so both have a reset value of “X.” In other words, there is no reset value for the ARP bit and the OVM bit.

Table 4–5. Status Register 0 (ST0) Bit Summary

Bit	Name	Reset value	Function
15–13	ARP	**X**	Auxiliary register pointer. These bits select the auxiliary register (AR) to be used in indirect addressing. When the ARP is loaded, the previous ARP value is copied to the auxiliary register buffer (ARB) in ST1. The ARP can be modified by memory-reference instructions when you use indirect addressing, and by the MAR or LST #0 instruction. When an LST #1 instruction is executed, the ARP is loaded with the same value as the ARB.
11	OVM	**X**	Overflow mode bit. This bit enables/disables the accumulator overflow saturation mode in the arithmetic logic unit (ALU). The OVM bit can be modified by the LST #0 instruction. OVM = 0 Disabled. An overflowed result is loaded into the accumulator without modification. The OVM bit can be cleared by the CLRC OVM instruction. OVM = 1 Overflow saturation mode. An overflowed result is loaded into the accumulator with either the most positive (00 7FFF FFFFh) or the most negative value (FF 8000 0000h). The OVM bit can be set by the SETC OVM instruction.

4–12 In Table 4–5, changed the reset value for the DP bit so it has a reset value of “X.” In other words, there is no reset value for the DP bit.

Table 4–5. Status Register 0 (ST0) Bit Summary (Continued)

Bit	Name	Reset value	Function
8–0	DP	**X**	Data memory page pointer bits. These bits specify the address of the current data memory page. The DP bits are concatenated with the 7 LSBs of an instruction word to form a direct memory address of 16 bits. The DP bits can be modified by the LST #0 or LDP instruction.

Page: **Change or Add:**

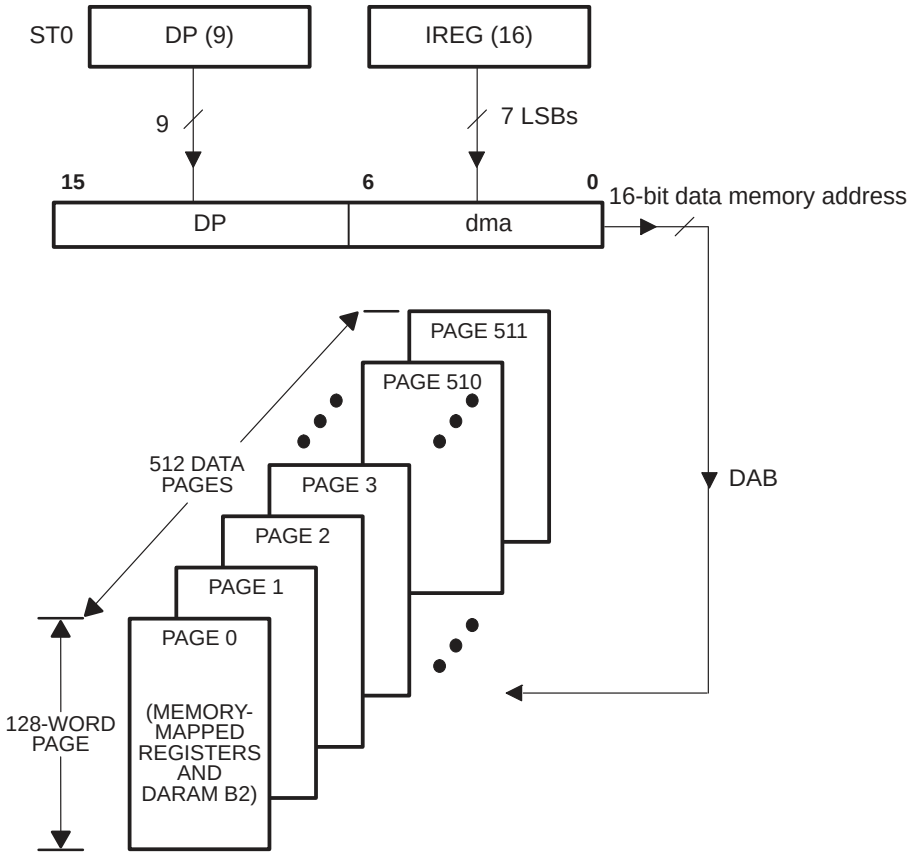
4–13 In Table 4–6, changed the reset value for the ARB bit and the TC bit so they have no reset value.

Table 4–6. Status Register 1 (ST1) Bit Summary

Bit	Name	Reset value	Function
15–13	ARB	**X**	Auxiliary register buffer. This 3-bit field holds the previous value contained in the auxiliary register pointer (ARP) in ST0. Whenever the ARP is loaded, the previous ARP value is copied to the ARB, except when using the LST #0 instruction. When the ARB is loaded using the LST #1 instruction, the same value is also copied to the ARP. This is useful when restoring context (when not using the automatic context save) in a subroutine that modifies the current ARP.
11	TC	**X**	Test/control flag bit. This 1-bit flag stores the results of the arithmetic logic unit (ALU) or parallel logic unit (PLU) test bit operations. The TC bit is affected by the APL, BIT, BITT, CMPR, CPL, NORM, OPL, and XPL instructions. The status of the TC bit determines if the conditional branch, call, and return instructions execute. The TC bit can be modified by the LST #1 instruction.

Page:	Change or Add:
5–2	In Figure 5–1, changed the page 0 length to “128-WORD PAGE.”

Figure 5–1. Direct Addressing



Page:	Change or Add:
5–22	In Example 5–13, added two new lines at the beginning of the example.

Example 5–13. Circular Addressing

```
mar    *,ar6
ldp    #,0

splk   #200h,CBSR1   ;  Circular buffer start register
splk   #203h,CBER1   ;  Circular buffer end register
splk   #0Eh,CBCR     ;  Enable AR6 pointing to buffer 1

lar    ar6,#200h     ;  Case 1
lacc   *             ;  AR6 = 200h

lar    ar6,#203h     ;  Case 2
lacc   *             ;  AR6 = 203h

lar    ar6,#200h     ;  Case 3
lacc   *+            ;  AR6 = 201h

lar    ar6,#203h     ;  Case 4
lacc   *+            ;  AR6 = 200h

lar    ar6,#200h     ;  Case 5
lacc   *–            ;  AR6 = 1FFh

lar    ar6,#203h     ;  Case 6
lacc   *–            ;  AR6 = 200h

lar    ar6,#202h     ;  Case 7
adrk   2             ;  AR6 = 204h

lar    ar6,#203h     ;  Case 8
adrk   2             ;  AR6 = 200h
```

Page:	Change or Add:
6–32	Changed the second operand for the ADD instruction.
Operands	0 ≤ shift ≤**16** (defaults to 0)
6–44	Changed the fourth operand for the AND instruction.
Operands	0 ≤ shift ≤ **16**
6–83	Changed the operand for the BSAR instruction.
Operands	1 ≤ shift ≤ **16**
6–85	Changed the description for the CALAD instruction.
Description	**The current program counter (PC) is incremented by 3 and pushed onto the top of the stack (TOS).** **Then, the one 2-word instruction or two 1-word instructions following the CALAD instruction are fetched from program memory and executed before the call is executed.** **Then, the contents of the accumulator low byte (ACCL) are loaded into the PC. Execution continues at this address.** **The CALAD instruction is used to perform computed subroutine calls. CALAD is a branch and call instruction (see Table 6–8).**
6–87	Changed the description for the CALLD instruction.
Description	**The current program counter (PC) is incremented by 4 and pushed onto the top of the stack (TOS).** **Then, the one 2-word instruction or two 1-word instructions following the CALLD instruction are fetched from program memory and executed before the call is executed.** **Then, the program memory address (pma) is loaded into the PC. Execution continues at this address. The current auxiliary register (AR) and auxiliary register pointer (ARP) are modified as specified. The pma can be either a symbolic or numeric address.** **CALLD is a branch and call instruction (see Table 6–8).**

Page: **Change or Add:**

6–91 Changed the description for the CCD instruction.

Description

If the specified conditions are met, the current program counter (PC) is incremented by 4 and pushed onto the top of the stack (TOS).

Then, the one 2-word instruction or two 1-word instructions following the CCD instruction are fetched from program memory and executed before the call is executed.

Then, the program memory address (pma) is loaded into the PC. Execution continues at this address. The pma can be either a symbolic or numeric address. Not all combinations of the conditions are meaningful. In addition, the NTC, TC, and BIO conditions are mutually exclusive.

If the specified conditions are not met, control is passed to the next instruction.

The CCD functions in the same manner as the CALLD instruction (page 6–87) if all conditions are true. CCD is a branch and call instruction (see Table 6–8).

6–103 Changed the opcode for the CRLT instruction to reflect the new values for bits 2, 1, and 0.

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	**1**	**0**	**0**

6–115 Changed the third operand for the LACC instruction.

Operands 0 ≤ shift ≤ **16** (defaults to 0)

6–127 Changed the table Cycles for a Single Instruction (short immediate addressing).

Cycles for a Single Instruction (short immediate addressing)

Operand	ROM	DARAM	SARAM	External Memory
	2	2	2	2+p_{code}

6–129 Changed the table Cycles for a Single Instruction (short immediate addressing).

Cycles for a Single Instruction (short immediate addressing)

Operand	ROM	DARAM	SARAM	External Memory
	2	2	2	2+p_{code}

6–188 Changed the fourth operand for the OR instruction.

Operands 0 ≤ shift ≤ **16**

Page: **Change or Add:**

6–261 Changed the second operand for the SUB instruction.

Operands 0 ≤ shift ≤ **16** (defaults to 0)

6–278 Changed the data memory address in Example 1 from 1905h to 1005h.

6–282 Changed the fourth operand for the XOR instruction.

Operands 0 ≤ shift ≤ **16**

8–6 In Figure 8–6, changed the word Off-chip to Reserved on the Program memory map for the range from 0040h to 8000h.

8–11 In Table 8–6, changed the values in the Off-Chip column for the first and fifth rows.

Table 8–6. 'C57S Program Memory Configuration

Bit values			ROM (2K-words)	SARAM (6K-words)	DARAM B0 (512-words)	Off-Chip
CNF	RAM	MP/$\overline{MC}$				
0	0	0	0000–07FF	Off-chip	Off-chip	**8000**–FFFF
1	0	0	0000–07FF	Off-chip	FE00–FFFF	**8000**–FDFF

8–32 Changed the last sentence in the fourth bullet.

- ❑ 32K words of global data memory are enabled initially in data spaces 8000h to FFFFh. After the code transfer is complete, the global memory is disabled before control is transferred to the destination address **in program memory**.

9–10 In Table 9–4, changed the sentences after Soft=0 and Soft=1. Also, add a sentence to the TSS register.

Table 9–4. Timer Control Register (TCR) Bit Summary

Bit	Name	Reset value	Function
11	Soft	0	This bit is used in conjunction with the Free bit to determine the state of the timer when a halt is encountered. When the Free bit is cleared, the Soft bit selects the emulation mode. Soft = 0 The timer stops immediately. Soft = 1 The timer stops **after decrementing to zero**.
4	TSS	0	Timer stop status bit. This bit stops or starts the on-chip timer. At reset, the TSS bit is cleared and the timer immediately starts timing. **Note that due to timer logic implementation, two successive writes of one to the TSS bit are required to properly stop the timer.**

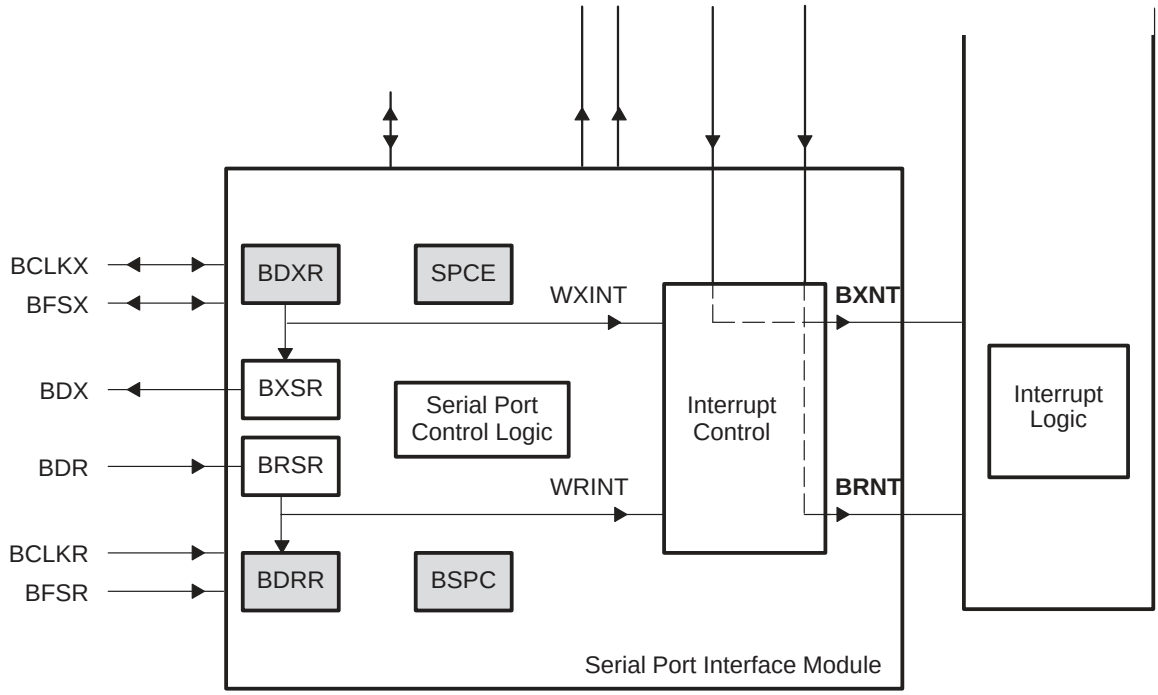
Page: **Change or Add:**

9–11 Deleted the last sentence in the Notes section and replace it with the sentence indicated.

The current value in the timer can be read by reading the TIM; the PSC can be read by reading the TCR. Because it takes two instructions to read both registers, there may be a change between the two reads as the counter decrements. Therefore, when making precise timing measurements, it may be more accurate to stop the timer to read these two values. **Due to timer logic implementation, two instructions are also required to properly stop the timer; therefore, two successive writes of one to the TSS bit should be made when the timer must be stopped.**

9–62 Changed the XINT and RINT labels found in the lower right portion of Figure 9–35.

Figure 9–35. ABU Block Diagram

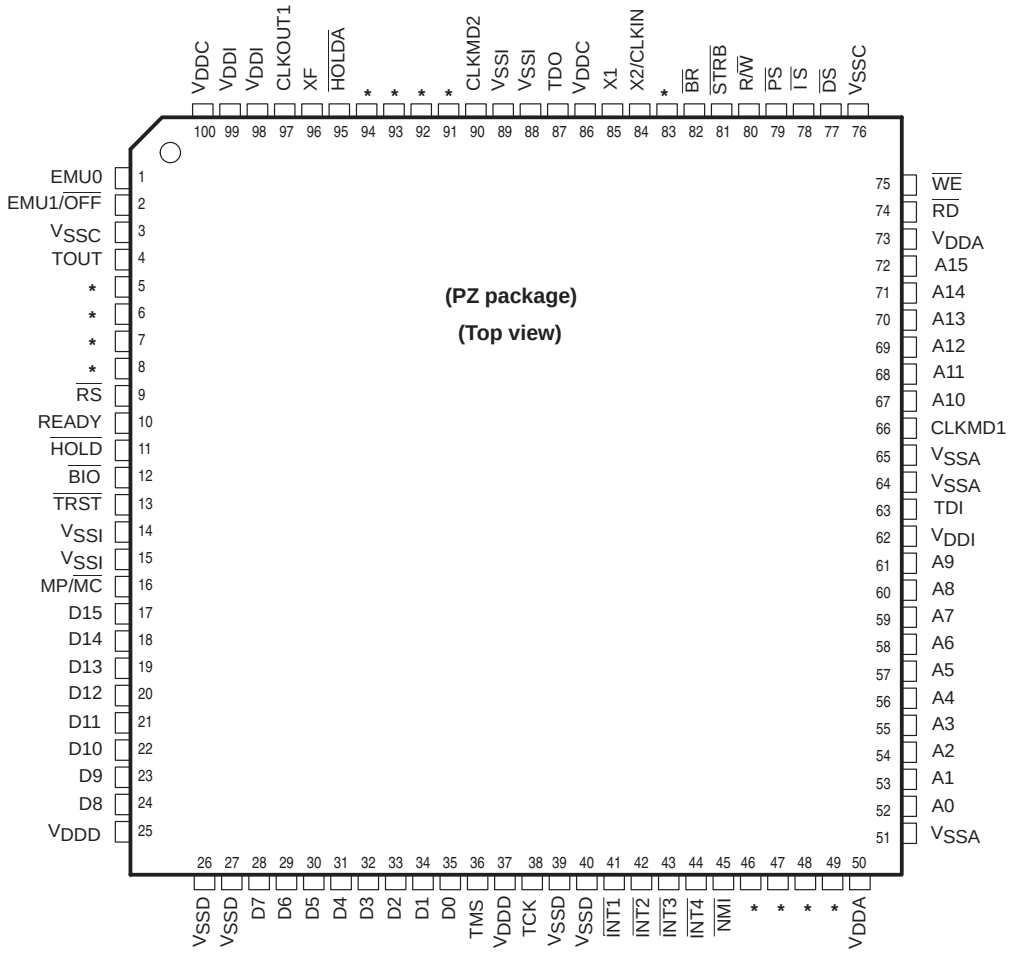


Page:	Change or Add:
9–63	Changed the last sentence in the first paragraph.

The internal ’C5X memory used for autobuffering consists of a 2K-word block of single-access memory that can be configured as data, program, or both (as with other single-access memory blocks). This memory can also be used by the CPU as general purpose storage, however, this is the only memory block in which autobuffering can occur. Since the BSP is implemented on several different TMS320 devices, the actual base address of the ABU memory may not be the same in all cases. **The 2K-word block of BSP memory is located at 800h–FFFh in data memory or at 8000h–87FFh in program memory as specified by the RAM and OVLY control bits**.

Page:	Change or Add:
A–4	In Figure A–2, changed the signal name on pin 80 to R/$\overline{W}$.

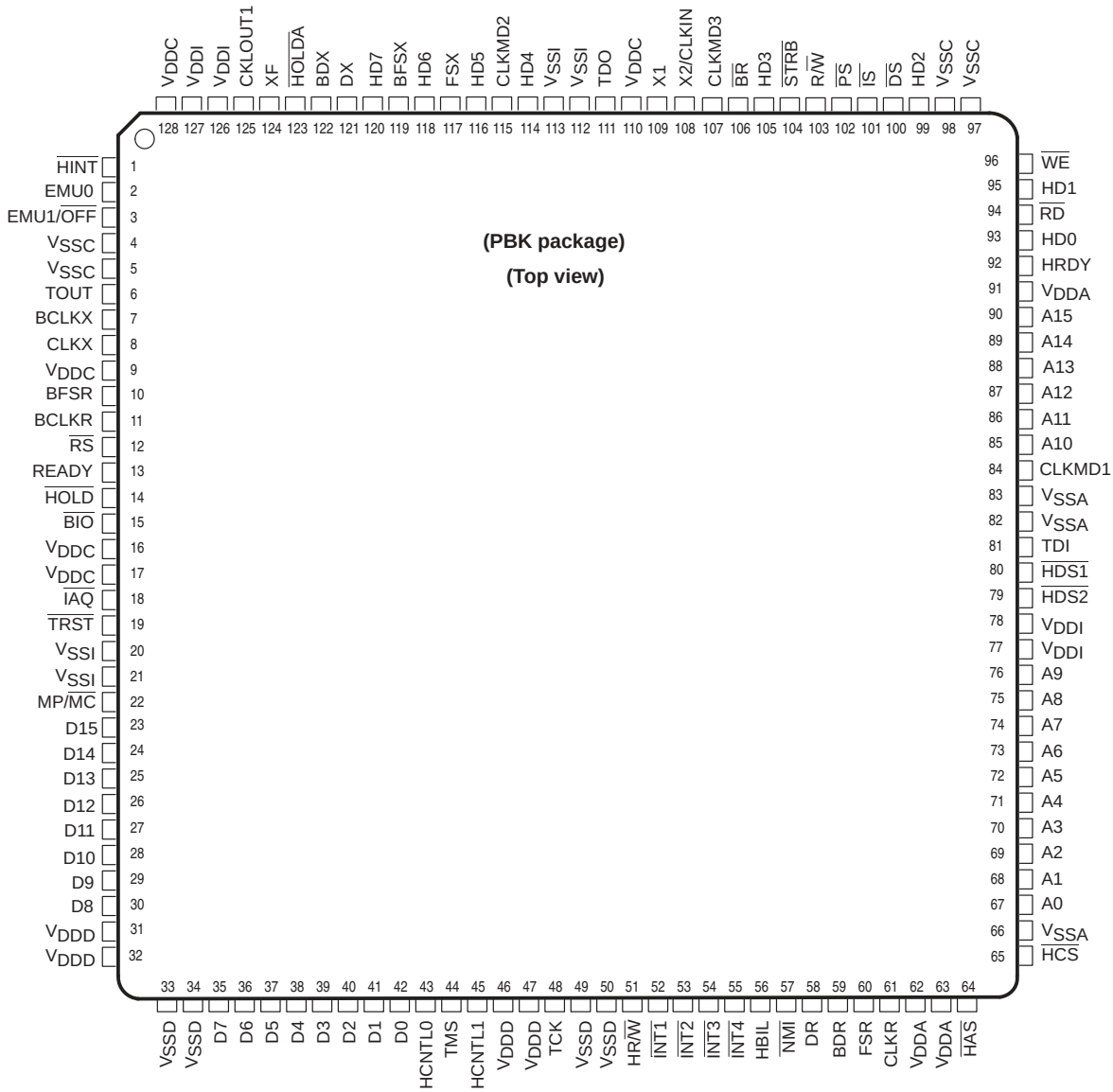
Figure A-2. Pin/Signal Assignments for the ’C51, ’C52, ’C53S, and ’LC56 in 100-Pin TQFP



Note: * These pins are reserved for specific devices (see Table A–6 on page A-12).

Page:	Change or Add:
A–6	In Figure A–3, changed the signal name on pin 108 to X2/CLKIN.

Figure A–3. Pin/Signal Assignments for the ’LC57 in 128-Pin TQFP



Page: **Change or Add:**

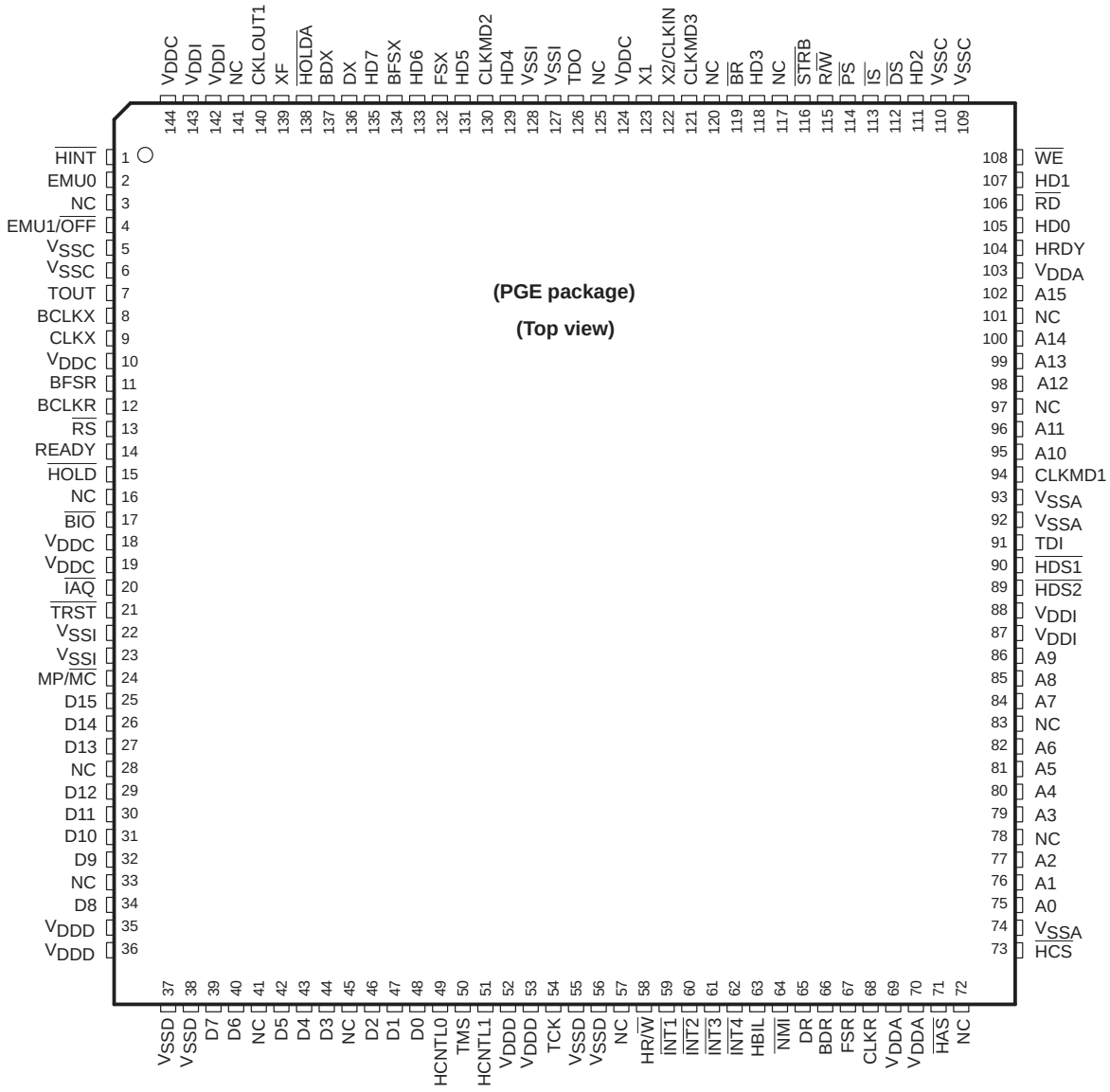
A–7 In Table A–3, changed the signal name on pin 108 to X2/CLKIN and reorder the signal names.

Table A–3. Signal/Pin Assignments for the ’LC57 in 128-Pin TQFP

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	67	CLKMD3	107	FSX	117	$\overline{IS}$	101	V_{DDD}	47
A1	68	CLKOUT1	125	$\overline{HAS}$	64	MP/$\overline{MC}$	22	V_{DDI}	77
A2	69	CLKR	61	HBIL	56	$\overline{NMI}$	57	V_{DDI}	78
A3	70	CLKX	8	HCNTL0	43	$\overline{PS}$	102	V_{DDI}	126
A4	71	D0	42	HCNTL1	45	$\overline{RD}$	94	V_{DDI}	127
A5	72	D1	41	$\overline{HCS}$	65	READY	13	V_{SSA}	66
A6	73	D2	40	HD0	93	$\overline{RS}$	12	V_{SSA}	82
A7	74	D3	39	HD1	95	R/$\overline{W}$	103	V_{SSA}	83
A8	75	D4	38	HD2	99	$\overline{STRB}$	104	V_{SSC}	4
A9	76	D5	37	HD3	105	TCK	48	V_{SSC}	5
A10	85	D6	36	HD4	114	TDI	81	V_{SSC}	97
A11	86	D7	35	HD5	116	TDO	111	V_{SSC}	98
A12	87	D8	30	HD6	118	TMS	44	V_{SSD}	33
A13	88	D9	29	HD7	120	TOUT	6	V_{SSD}	34
A14	89	D10	28	$\overline{HDS1}$	80	$\overline{TRST}$	19	V_{SSD}	49
A15	90	D11	27	$\overline{HDS2}$	79	V_{DDC}	9	V_{SSD}	50
BCLKR	11	D12	26	$\overline{HINT}$	1	V_{DDA}	91	V_{SSI}	20
BCLKX	7	D13	25	$\overline{HOLD}$	14	V_{DDA}	63	V_{SSI}	21
BDR	59	D14	24	$\overline{HOLDA}$	123	V_{DDA}	62	V_{SSI}	112
BDX	122	D15	23	HRDY	92	V_{DDC}	16	V_{SSI}	113
BFSR	10	DR	58	HR/$\overline{W}$	51	V_{DDC}	17	$\overline{WE}$	96
BFSX	119	$\overline{DS}$	100	$\overline{IAQ}$	18	V_{DDC}	110	X1	109
$\overline{BIO}$	15	DX	121	$\overline{INT1}$	52	V_{DDC}	128	X2/CLKIN	108
$\overline{BR}$	106	EMU0	2	$\overline{INT2}$	53	V_{DDD}	31	XF	124
CLKMD1	84	EMU1/$\overline{OFF}$	3	$\overline{INT3}$	54	V_{DDD}	32		
CLKMD2	115	FSR	60	$\overline{INT4}$	55	V_{DDD}	46		

Page:	Change or Add:
A–10	In Figure A–5, corrected the signal names for pins 1–16, 28–45, 57–71, and 78–141; changed the signal name on pin 122 to X2/CLKIN.

Figure A–5. Pin/Signal Assignments for the ’C57S in 144-Pin TQFP



Note: NC These pins are not connected (reserved).

Page:	Change or Add:
A–11	In Table A–5, corrected the signal names for pins 1–16, 28–45, 57–71, and 78–141; changed the signal name on pin 122 to X2/CLKIN; reordered the signal names.

Table A–5. Signal/Pin Assignments for the 'C57S in 144-Pin TQFP

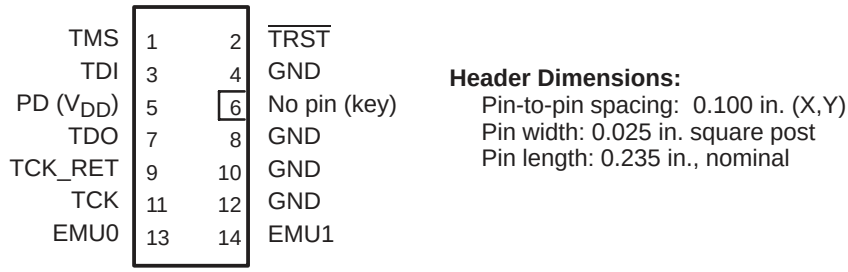
Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A0	75	CLKX	9	HD0	105	TCK	54	V_{SSD}	37
A1	76	D0	48	HD1	107	TDI	91	V_{SSD}	38
A2	77	D1	47	HD2	111	TDO	126	V_{SSD}	55
A3	79	D2	46	HD3	118	TMS	50	V_{SSD}	56
A4	80	D3	44	HD4	129	TOUT	7	V_{SSI}	22
A5	81	D4	43	HD5	131	$\overline{TRST}$	21	V_{SSI}	23
A6	82	D5	42	HD6	133	V_{DDA}	69	V_{SSI}	127
A7	84	D6	40	HD7	135	V_{DDA}	70	V_{SSI}	128
A8	85	D7	39	$\overline{HDS1}$	90	V_{DDA}	103	$\overline{WE}$	108
A9	86	D8	34	$\overline{HDS2}$	89	V_{DDC}	10	X1	123
A10	95	D9	32	$\overline{HINT}$	1	V_{DDC}	18	X2/CLKIN	122
A11	96	D10	31	$\overline{HOLD}$	15	V_{DDC}	19	XF	139
A12	98	D11	30	$\overline{HOLDA}$	138	V_{DDC}	124	†	3
A13	99	D12	29	HRDY	104	V_{DDC}	144	†	16
A14	100	D13	27	HR/$\overline{W}$	58	V_{DDD}	35	†	28
A15	102	D14	26	$\overline{IAQ}$	20	V_{DDD}	36	†	33
BCLKR	12	D15	25	$\overline{INT1}$	59	V_{DDD}	52	†	41
BCLKX	8	DR	65	$\overline{INT2}$	60	V_{DDD}	53	†	45
BDR	66	$\overline{DS}$	112	$\overline{INT3}$	61	V_{DDI}	87	†	57
BDX	137	DX	136	$\overline{INT4}$	62	V_{DDI}	88	†	72
BFSR	11	EMU0	2	$\overline{IS}$	113	V_{DDI}	142	†	78
BFSX	134	EMU1/$\overline{OFF}$	4	MP/$\overline{MC}$	24	V_{DDI}	143	†	83
$\overline{BIO}$	17	FSR	67	$\overline{NMI}$	64	V_{SSA}	74	†	97
$\overline{BR}$	119	FSX	132	$\overline{PS}$	114	V_{SSA}	92	†	101
CLKMD1	94	$\overline{HAS}$	71	$\overline{RD}$	106	V_{SSA}	93	†	117
CLKMD2	130	HBIL	63	READY	14	V_{SSC}	5	†	120
CLKMD3	121	HCNTL0	49	$\overline{RS}$	13	V_{SSC}	6	†	125
CLKOUT1	140	HCNTL1	51	R/$\overline{W}$	115	V_{SSC}	109	†	141
CLKR	68	$\overline{HCS}$	73	$\overline{STRB}$	116	V_{SSC}	110		

† These pins are not connected (reserved).

Page: **Change or Add:**

D–2 In Figure D–1, changed the PD pin 5 from +5V to V_{DD}.

Figure D–1. Header Signals and Header Dimensions



In Table D–1, changed the voltage for pin 5 (the PD pin) from +5V to V_{DD}.

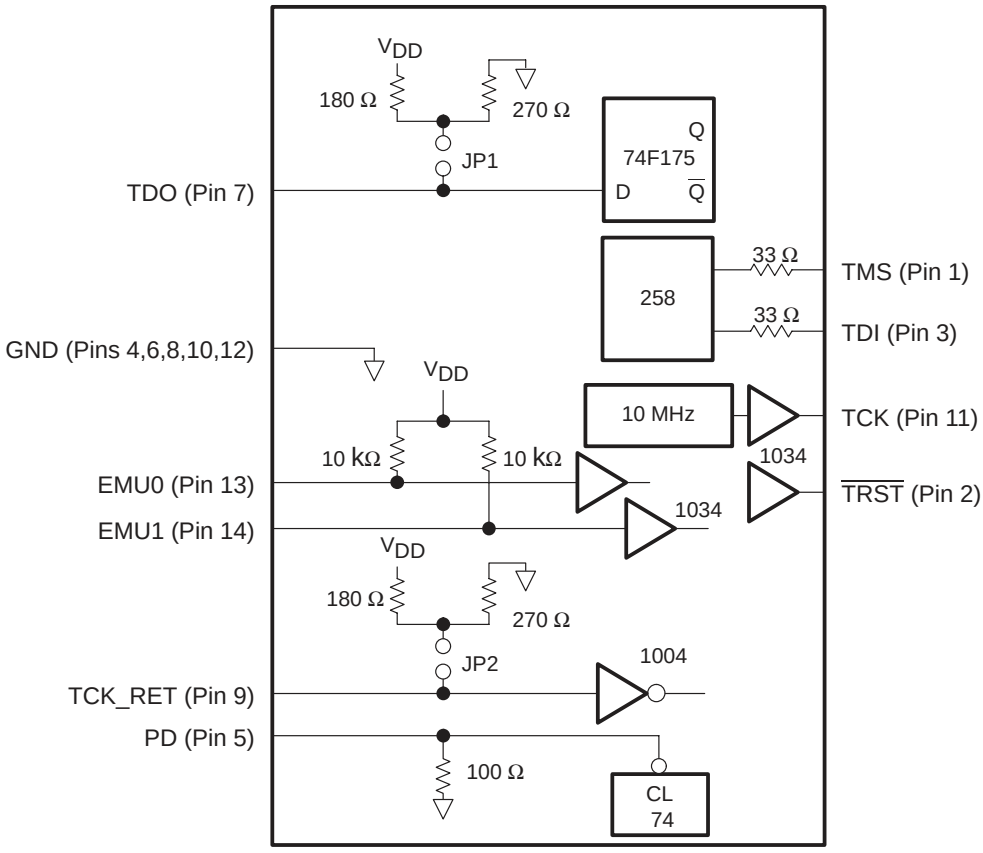
Table D–1. XDS510 Header Signal Description

Pin	Signal	State	Target State	Description
5	PD	I	O	Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD should be tied to **V_{DD}** in the target system.

Page: **Change or Add:**

D–5 In Figure D–2, changed the voltages from +5V to V_{DD}.

Figure D–2. Emulator Cable Pod Interface

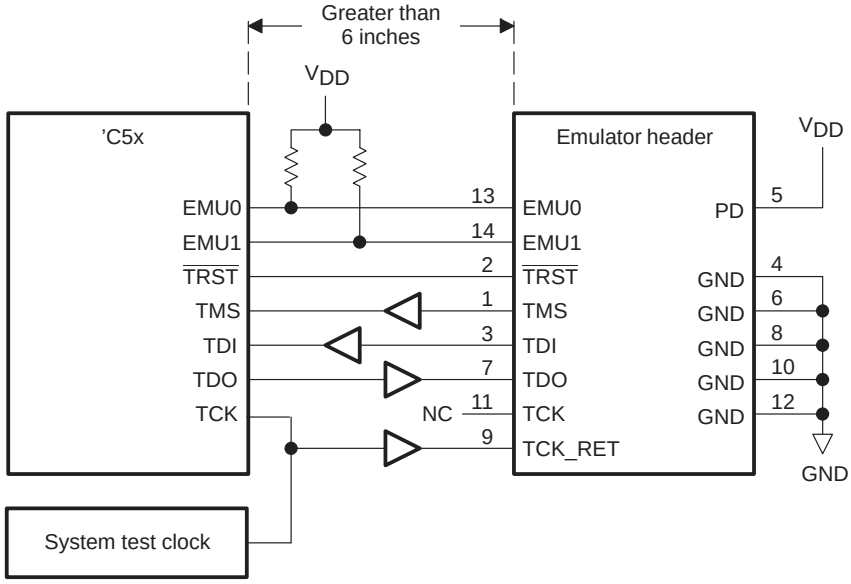


NOTE:
All devices are 74AS, unless otherwise specified.

Page: **Change or Add:**

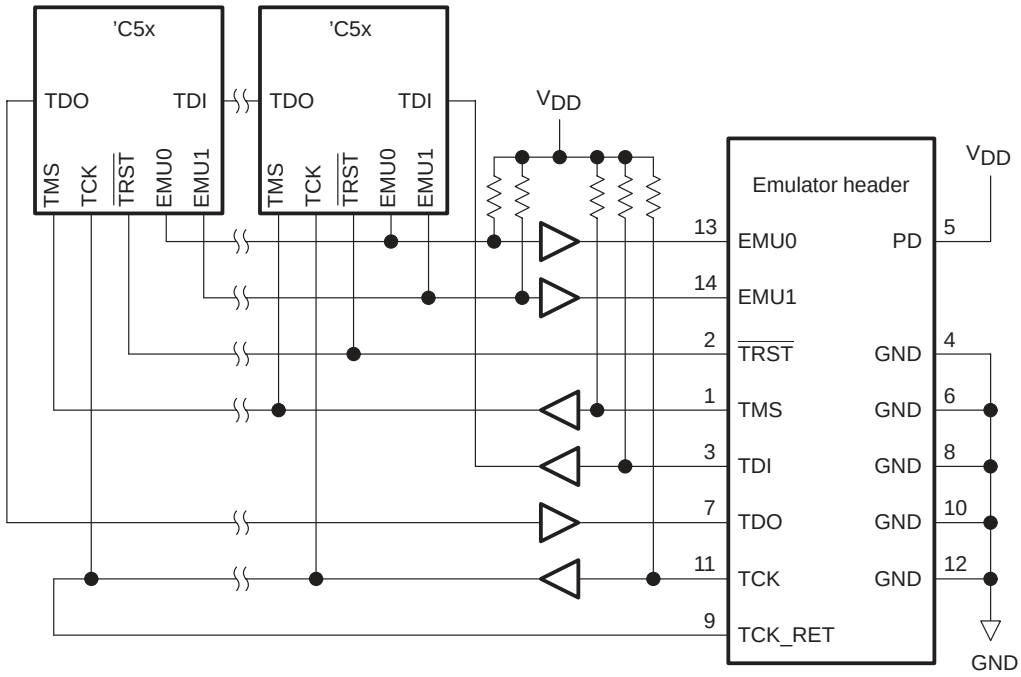
D–7 In Figure D–4, changed the voltages from +5V to V_{DD}.

Figure D–4. Target-System Generated Test Clock



D–8 In Figure D–5, changed the voltages from +5V to V_{DD}.

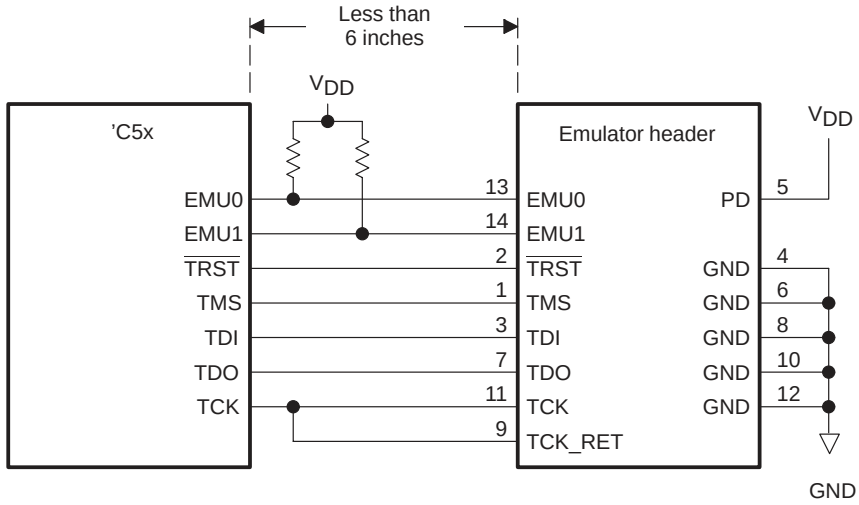
Figure D–5. Multiprocessor Connections



Page: **Change or Add:**

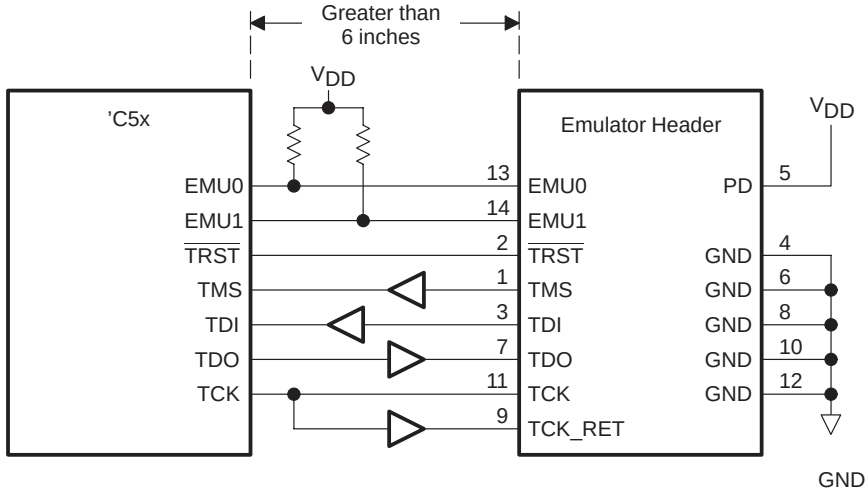
D–9 In Figure D–6, changed the voltages from +5V to V_{DD}.

Figure D–6. Emulator Connections Without Signal Buffering



D–10 In Figure D–7, change the voltages from +5V to V_{DD}.

Figure D–7. Buffered Signals



Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

Z