TMS320C20x User's Guide

Literature Number: SPRU127C
April 1999

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products 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, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not 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. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Copyright © 1999, Texas Instruments Incorporated

Preface

Read This First

About This Manual

This user's guide describes the architecture, hardware, assembly language instructions, and general operation of the TMS320C20x† digital signal processors (DSPs). This manual can also be used as a reference guide for developing hardware and/or software applications. In this document, 'C20x† refers to any of the TMS320C20x devices, except where device-specific information is explicitly stated. When device-specific information is given, the device name may be abbreviated; for example, TMS320C203 will be abbreviated as 'C203. This manual covers 'C203, 'LC203, 'C206, 'LC206, and 'F206 devices. For pinouts, electrical characteristics, and timing diagrams, refer to the data sheets of the individual devices.

How to Use This Manual

Chapter 1, *Introduction*, summarizes the TMS320 family of products and then introduces the key features of the TMS320C20x generation of that family. Chapter 2, *Architectural Overview*, summarizes the 'C20x architecture, providing information about the CPU, bus structure, memory, on-chip peripherals, and scanning logic.

If you are reading this manual to learn about the 'C209, Chapter 11 is important for you. There are some notable differences between the 'C209 and other 'C20x devices, and Chapter 11 explains these differences. In addition, it shows how to use this manual to get a complete picture of the 'C209.

The following table points you to major topics.

† The generic name '2xx refers to all DSPs using the 2xLP DSP core. This user guide revision uses '20x, a subset of '2xx, to specifically reference the 'C/LC203, 'F206, and the C/LC206.

For this information:	Look here:
Addressing modes (for addressing data memory)	Chapter 6, *Addressing Modes*
Assembly language instructions	Chapter 7, *Assembly Language Instructions*
Assembly language instructions of TMS320C1x, ’C2x, ’C20x, and ’C5x compared	Appendix C, *TMS320C1x/C2x/C2xx/C5x Instruction Set Comparison*
Boot loader	Chapter 4, *Memory and I/O Spaces*
Clock generator	Chapter 8, *On-Chip Peripherals*
CPU	Chapter 3, *Central Processing Unit*
Custom ROM from TI	Appendix E, *Submitting ROM Codes to TI*
Emulator	Appendix F, *Design Considerations for Using XDS510 Emulator*
Features	Chapter 1, *Introduction* Chapter 2, *Architectural Overview*
Input/output ports	Chapter 4, *Memory and I/O Spaces*
Interrupts	Chapter 5, *Program Control*
Memory configuration	Chapter 4, *Memory and I/O Spaces*
Memory interfacing	Chapter 4, *Memory and I/O Spaces*
On-chip peripherals	Chapter 8, *On-Chip Peripherals*
Pipeline	Chapter 5, *Program Control*
Program control	Chapter 5, *Program Control*
Program examples	Appendix D, *Program Examples*
Program-memory address generation	Chapter 5, *Program Control*
Registers summarized	Appendix A, *Register Summary*
Serial ports	Chapter 9, *Synchronous Serial Port* Chapter 10, *Asynchronous Serial Port*
Stack	Chapter 5, *Program Control*
Status registers	Chapter 5, *Program Control*
Timer	Chapter 8, *On-Chip Peripherals*
TMS320C209 differences and similarities	Chapter 11, *TMS320C209*
Wait-state generator	Chapter 8, *On-Chip Peripherals*

Notational Conventions

This document uses the following conventions:

- Program listings and program examples are shown in a `special typeface`.

  Here is a segment of a program listing:

  ```
  OUTPUT LDP     #6          ;select data page 6
         BLDD    #300, 20h   ;move data at address 300h to 320h
         RET
  ```

- In syntax descriptions, **bold** portions of a syntax should be entered as shown; *italic* portions of a syntax identify information that you specify. Here is an example of an instruction syntax:

  BLDD *source, destination*

  BLDD is the instruction mnemonic, which must be typed as shown. You specify the two parameters, *source* and *destination*.

- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you do not type the brackets themselves. You separate each optional operand from required operands with a comma and a space. Here is a sample syntax:

  BLDD *source*, *destination* [, **AR***n*]

  BLDD is the instruction. The two required operands are *source* and *destination*, and the optional operand is **AR***n*. **AR** is bold and *n* is italic; if you choose to use **AR***n*, you must type the letters A and R and then supply a chosen value for *n* (in this case, a value from 0 to 7). Here is an example:

  ```
  BLDD *, #310h, AR3
  ```

Information About Cautions

This book contains cautions.

CAUTION

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 carefully.

Related Documentation From Texas Instruments

This section describes related TI™ documents that can be ordered by calling the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the document by its title and literature number.

The following data sheets contain the electrical and timing specifications for the TMS320C20x devices, as well as signal descriptions and pinouts for all of the available packages:

- TMS320C20x data sheets (literature number SPRS025 and SPRS065)
- TMS320F20x data sheet (literature number SPRS050). This data sheet covers the TMS320F20x devices that have on-chip flash memory.

The books listed below provide additional information about using the TMS320C2xx devices and related support tools, as well as more general information about using the TMS320 family of DSPs.

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.

TMS320C2xx PC Emulator Installation Guide (literature number SPRU152) describes the installation of the XDS510 PC emulator and the C source debugger for OS/2 and MS-Windows operating systems.

TMS320C2xx C Source Debugger User’s Guide (literature number SPRU151) tells you how to invoke the ’C2xx emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS320C2xx Simulator Getting Started (literature number SPRU137) describes how to install the TMS320C2xx simulator and the C source debugger for the 'C2xx. The installation for MS-DOS™, PC-DOS™, SunOS™, Solaris™, and HP-UX™ systems is covered.

TMS320C2xx Emulator Getting Started Guide (literature number SPRU209) tells you how to install the Windows™ 3.1 and Windows™ 95 versions of the 'C2xx emulator and C source debugger interface.

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 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.

Digital Signal Processing Applications with the TMS320 Family, Volumes 1, 2, and 3 (literature numbers SPRA012, SPRA016, SPRA017) Volumes 1 and 2 cover applications using the 'C10 and 'C20 families of fixed-point processors. Volume 3 documents applications using both fixed-point processors as well as the 'C30 floating-point processor.

TMS320 DSP Designer's Notebook: Volume 1 (literature number SPRT125)***.*** Presents solutions to common design problems using 'C2x, 'C3x, 'C4x, 'C5x, and other TI DSPs***.***

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.

Related Articles

"A Greener World Through DSP Controllers", Panos Papamichalis, *DSP & Multimedia Technology*, September 1994.

"A Single-Chip Multiprocessor DSP for Image Processing—TMS320C80", Dr. Ing. Dung Tu, *Industrie Elektronik*, Germany, March 1995.

"Application Guide with DSP Leading-Edge Technology", Y. Nishikori, M. Hattori, T. Fukuhara, R.Tanaka, M. Shimoda, I. Kudo, A.Yanagitani, H. Miyaguchi, et al., *Electronics Engineering*, November 1995.

"Approaching the No-Power Barrier", Jon Bradley and Gene Frantz, *Electronic Design*, January 9, 1995.

"Beware of BAT: DSPs Add Brilliance to New Weapons Systems", Panos Papamichalis, *DSP & Multimedia Technology*, October 1994.

"Choose DSPs for PC Signal Processing", Panos Papamichalis, *DSP & Multimedia Technology*, January/February 1995.

"Developing Nations Take Shine to Wireless", Russell MacDonald, Kara Schmidt and Kim Higden, *EE Times*, October 2, 1995.

"Digital Signal Processing Solutions Target Vertical Application Markets", Ron Wages, *ECN*, September 1995.

"Digital Signal Processors Boost Drive Performance", Tim Adcock, *Data Storage*, September/October 1995.

"DSP and Speech Recognition, An Origin of the Species", Panos Papamichalis, *DSP & Multimedia Technology*, July 1994.

"DSP Design Takes Top-Down Approach", Andy Fritsch and Kim Asal, *DSP Series Part III*, *EE Times*, July 17, 1995.

"DSPs Advance Low-Cost 'Green' Control", Gregg Bennett, *DSP Series Part II*, *EE Times*, April 17, 1995.

"DSPs Do Best on Multimedia Applications", Doug Rasor, *Asian Computer World*, October 9–16, 1995.

"DSPs: Speech Recognition Technology Enablers", Gene Frantz and Gregg Bennett, *I&CS*, May 1995.

"Easing JTAG Testing of Parallel-Processor Projects", Tony Coomes, Andy Fritsch, and Reid Tatge, *Asian Electronics Engineer*, Manila, Philippines, November 1995.

"Fixed or Floating? A Pointed Question in DSPs", Jim Larimer and Daniel Chen, *EDN*, August 3, 1995.

"Function-Focused Chipsets: Up the DSP Integration Core", Panos Papamichalis, *DSP & Multimedia Technology*, March/April 1995.

"GSM: Standard, Strategien und Systemchips", Edgar Auslander, *Elektronik Praxis*, Germany, October 6, 1995.

"High Tech Copiers to Improve Images and Reduce Paperwork", Karl Guttag, *Document Management*, July/August 1995.

"Host-Enabled Multimedia: Brought to You by DSP Solutions", Panos Papamichalis, *DSP & Multimedia Technology*, September/October 1995.

"Integration Shrinks Digital Cellular Telephone Designs", Fred Cohen and Mike McMahan, *Wireless System Design*, November 1994.

"On-Chip Multiprocessing Melds DSPs", Karl Guttag and Doug Deao, *DSP Series Part III*, *EE Times*, July 18, 1994.

"Real-Time Control", Gregg Bennett, *Appliance Manufacturer*, May 1995.

"Speech Recognition", P.K. Rajasekaran and Mike McMahan, *Wireless Design & Development*, May 1995.

"Telecom Future Driven by Reduced Milliwatts per DSP Function", Panos Papamichalis, *DSP & Multimedia Technology*, May/June 1995.

"The Digital Signal Processor Development Environment", Greg Peake, *Embedded System Engineering*, United Kingdom, February 1995.

"The Growing Spectrum of Custom DSPs", Gene Frantz and Kun Lin, *DSP Series Part II*, *EE Times*, April 18, 1994.

"The Wide World of DSPs, " Jim Larimer, *Design News*, June 27, 1994.

"Third-Party Support Drives DSP Development for Uninitiated and Experts Alike", Panos Papamichalis, *DSP & Multimedia Technology*, December 1994/January 1995.

"Toward an Era of Economical DSPs", John Cooper, *DSP Series Part I*, *EE Times*, Jan. 23, 1995.

Trademarks

TI, 320 Hotline On-line, XDS510, XDS510PP, XDS510WS, and XDS511 are trademarks of Texas Instruments Incorporated.

HP-UX is a trademark of Hewlett-Packard Company.

Intel is a trademark of Intel Corporation.

MS-DOS and Windows are registered trademarks of Microsoft Corporation.

PAL® is a registered trademark of Advanced Micro Devices, Inc.

OS/2, PC, and PC-DOS are trademarks of International Business Machines Corporation.

Solaris and SunOS are trademarks of Sun Microsystems, Inc.

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		Email: dsph@ti.com

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

1 Introduction **1-1**

Summarizes the features of the TMS320 family of products and presents typical applications. Describes the TMS320C20x DSP and lists its key features.

1.1 TMS320 Family 1-2
1.2 TMS320C20x Generation 1-4
1.3 Key Features of the TMS320C20x 1-5

2 Architectural Overview **2-1**

Summarizes the TMS320C20x architecture. Provides information about the CPU, bus structure, memory, on-chip peripherals, and scanning logic.

2.1 ’C20x Bus Structure 2-3
2.2 Central Processing Unit 2-5
2.3 Memory and I/O Spaces 2-7
2.4 Program Control 2-10
2.5 On-Chip Peripherals 2-11
2.6 Scanning-Logic Circuitry 2-13

3 Central Processing Unit **3-1**

Describes the TMS320C20x CPU. Includes information about the central arithmetic logic unit, the accumulator, the shifters, the multiplier, and the auxiliary register arithmetic unit. Concludes with a description of the status register bits.

3.1 Input Scaling Section 3-3
3.2 Multiplication Section 3-5
3.3 Central Arithmetic Logic Section 3-8
3.4 Auxiliary Register Arithmetic Unit (ARAU) 3-12
3.5 Status Registers ST0 and ST1 3-15

4 Memory and I/O Spaces **4-1**

Describes the configuration and use of the TMS320C20x memory and I/O spaces. Includes memory/address maps and descriptions of the HOLD (direct memory access) operation and the on-chip bootloader.

4.1 Overview of the Memory and I/O Spaces 4-2
4.2 Program Memory 4-5
4.3 Local Data Memory 4-7
4.4 Global Data Memory 4-11
4.5 I/O Space 4-14
4.6 Direct Memory Access Using the HOLD Operation 4-18
4.7 Device-Specific Information 4-22
4.8 ’C203 Bootloader 4-30
4.9 ’C206/LC206 Bootloader 4-39

5 Program Control **5-1**

Describes the TMS320C20x hardware and software features used in controlling program flow, including program-address generation logic and interrupts. Also describes the reset operation and power-down mode.

5.1 Program-Address Generation 5-2
5.2 Pipeline Operation 5-7
5.3 Branches, Calls, and Returns 5-8
5.4 Conditional Branches, Calls, and Returns 5-10
5.5 Repeating a Single Instruction 5-14
5.6 Interrupts 5-15
5.7 Reset Operation 5-35
5.8 Power-Down Mode 5-40

6 Addressing Modes **6-1**

Describes the operation and use of the TMS320C20x data-memory addressing modes.

6.1 Immediate Addressing Mode 6-2
6.2 Direct Addressing Mode 6-4
6.3 Indirect Addressing Mode 6-9

7 Assembly Language Instructions **7-1**

Describes the TMS320C20x assembly language instructions in alphabetical order. Begins with a summary of the TMS320C20x instructions.

7.1 Instruction Set Summary 7-2
7.2 How To Use the Instruction Descriptions 7-12
7.3 Instruction Descriptions 7-20

8 On-Chip Peripherals **8-1**

Introduces the TMS320C20x on-chip peripherals. Describes the clock generator, the CLKOUT1-pin control register, the timer, the wait-state generator, and the general-purpose I/O pins.

8.1 Control of On-Chip Peripherals 8-2
8.2 Clock Generator 8-4
8.3 CLKOUT1-Pin Control (CLK) Register 8-7
8.4 Timer 8-8
8.5 Wait-State Generator 8-15
8.6 General-Purpose I/O Pins 8-18

9 Synchronous Serial Port **9-1**

Describes the operation and control of the TMS320C20x on-chip synchronous serial port.

9.1 Overview of the Synchronous Serial Port 9-2
9.2 Components and Basic Operation 9-3
9.3 Controlling and Resetting the Port 9-8
9.4 Managing the Contents of the FIFO Buffers 9-15
9.5 Transmitter Operation 9-16

9.6 Receiver Operation 9-22
9.7 Troubleshooting 9-25
9.8 Enhanced Synchronous Serial Port (ESSP) 9-29
9.9 ESSP Pins 9-30
9.10 ESSP Registers 9-32
9.11 ESSP Register Programming Considerations 9-40

10 Asynchronous Serial Port 10-1

Describes the operation and control of the TMS320C20x on-chip asynchronous serial port.

10.1 Overview of the Asynchronous Serial Port 10-2
10.2 Components and Basic Operation 10-3
10.3 Controlling and Resetting the Port 10-7
10.4 Transmitter Operation 10-19
10.5 Receiver Operation 10-20

11 TMS320C209 11-1

Describes how the TMS320C209 differs from other TMS320C20x devices and is a central resource for all the TMS320C209-specific control registers and configuration information.

11.1 ’C209 Versus Other ’C20x Devices 11-2
11.2 ’C209 Memory and I/O Spaces 11-5
11.3 ’C209 Interrupts 11-10
11.4 ’C209 On-Chip Peripherals 11-15

A Register Summary A-1

Is a concise, central resource for information about the TMS320C20x on-chip registers. Includes addresses, reset values, and descriptive illustrations for the registers.

A.1 Addresses and Reset Values A-2
A.2 Register Descriptions A-4

B TMS320F206 Flash Serial Loader B-1

Discusses the TMS320F206 Flash Serial Loader.

B.1 TMS320F206 Flash Serial Loader Features B-2
B.2 Functional Description B-3
B.3 Serial Loader Code B-6

C TMS320C1x/C2x/C20x/C5x Instruction Set Comparison C-1

Discusses the compatibility of program code among the following devices: TMS320C1x, TMS320C2x, TMS320C20x, and TMS320C5x.

C.1 Using the Instruction Set Comparison Table C-2
C.2 Enhanced Instructions C-5
C.3 Instruction Set Comparison Table C-6

D Program Examples D-1

Presents examples of assembly language programs for the TMS320C20x, primarily examples for the on-chip peripherals.

D.1 About These Program Examples D-2
D.2 Shared Program Code D-5
D.3 Task-Specific Program Code D-8
D.4 Introduction to Generating Bootloader Code D-23

E Submitting ROM Codes to TI **E-1**

Explains the process for submitting custom program code to TI for designing masks for the on-chip ROM on a TMS320 DSP.

F Design Considerations for Using XDS510 Emulator **F-1**

Describes the JTAG emulator cable and how to construct a 14-pin connector on your target system and how to connect the target system to the emulator.

F.1 Designing Your Target System's Emulator Connector (14-Pin Header) F-2
F.2 Bus Protocol F-4
F.3 Emulator Cable Pod F-5
F.4 Emulator Cable Pod Signal Timing F-6
F.5 Emulation Timing Calculations F-7
F.6 Connections Between the Emulator and the Target System F-10
F.7 Physical Dimensions for the 14-Pin Emulator Connector F-14
F.8 Emulation Design Considerations F-16

G Glossary **G-1**

Explains terms, abbreviations, and acronyms used throughout this book.

Figures

2–1 Overall Block Diagram of the ’C20x 2-2
2–2 Bus Structure Block Diagram 2-4
3–1 Block Diagram of the Input Scaling, Central Arithmetic Logic, and Multiplication Sections of the CPU 3-2
3–2 Block Diagram of the Input Scaling Section 3-3
3–3 Operation of the Input Shifter for SXM = 0 3-4
3–4 Operation of the Input Shifter for SXM = 1 3-4
3–5 Block Diagram of the Multiplication Section 3-5
3–6 Block Diagram of the Central Arithmetic Logic Section 3-8
3–7 Shifting and Storing the High Word of the Accumulator 3-11
3–8 Shifting and Storing the Low Word of the Accumulator 3-11
3–9 ARAU and Related Logic 3-12
3–10 Status Register ST0 3-15
3–11 Status Register ST1 3-15
4–1 Interface With External Program Memory 4-6
4–2 Pages of Data Memory 4-7
4–3 Interface With External Local Data Memory 4-10
4–4 GREG Register Set to Configure 8K for Global Data Memory 4-12
4–5 Global and Local Data Memory for GREG = 11100000 4-12
4–6 Using 8000h–FFFFh for Local and Global External Memory 4-13
4–7 I/O Address Map for the ’C20x 4-14
4–8 I/O Port Interface Circuitry 4-17
4–9 $\overline{\text{HOLD}}$ Deasserted Before Reset Deasserted 4-20
4–10 Reset Deasserted Before $\overline{\text{HOLD}}$ Deasserted 4-21
4–11 ’C203 Address Map 4-23
4–12 TMS320C206, TMS320LC206 Memory Map Configurations 4-26
4–13 TMS320F206 Memory Map Configuration 4-28
4–14 PMST Register Selection for $\overline{\text{RD}}$ 4-29
4–15 Simplified Block Diagram of Bootloader Operation 4-30
4–16 Connecting the EPROM to the Processor 4-31
4–17 Storing the Program in the EPROM 4-33
4–18 Program Code Transferred From 8-Bit EPROM to 16-Bit RAM 4-35
4–19 Interrupt Vectors Transferred First During Boot Load 4-36
4–20 Program Memory Status (PMST) Register – (I/O space FFE4h) 4-40
4–21 Enhanced ’C206 Bootloader Options 4-42
4–22 Boot-load Flowchart 4-43

4–23 Destination Address Space for Programs in Program Memory 4-44
4–24 16-Bit Word Transfer 4-47
4–25 Host-’C206 Interface for SSP Boot-load Option 4-48
4–26 Figure 9. 8-Bit Word Transfer 4-49
4–27 16-Bit Source Address for Parallel EPROM Boot Mode 4-51
4–28 Handshake Protocol 4-53
4–29 16-Bit Entry Address for Warm-Boot Mode 4-54
5–1 Program-Address Generation Block Diagram 5-2
5–2 A Push Operation 5-5
5–3 A Pop Operation 5-6
5–4 4-Level Pipeline Operation 5-7
5–5 $\overline{\text{INT2}}$/$\overline{\text{INT3}}$ Request Flow Chart 5-18
5–6 Maskable Interrupt Operation Flow Chart 5-20
5–7 ’C20x Interrupt Flag Register (IFR) — Data-Memory Address 0006h 5-21
5–8 ’C20x Interrupt Mask Register (IMR) — Data-Memory Address 0004h 5-23
5–9 ’C20x Interrupt Control Register (ICR) — I/O-Space Address FFECh 5-26
5–10 Nonmaskable Interrupt Operation Flow Chart 5-29
5–11 Direct Addressing Context Save 5-33
5–12 Indirect Addressing Context Save 5-34
6–1 Instruction Register Contents for Example 6–1 6-2
6–2 Two Words Loaded Consecutively to the Instruction Register in Example 6–2 6-3
6–3 Pages of Data Memory 6-4
6–4 Instruction Register (IR) Contents in Direct Addressing Mode 6-5
6–5 Generation of Data Addresses in Direct Addressing Mode 6-5
6–6 Instruction Register Content in Indirect Addressing 6-12
7–1 Bit Numbers and Their Corresponding Bit Codes for BIT Instruction 7-45
7–2 Bit Numbers and Their Corresponding Bit Codes for BITT Instruction 7-47
7–3 LST #0 Operation 7-87
7–4 LST #1 Operation 7-88
8–1 Using the Internal Oscillator 8-4
8–2 Using an External Oscillator 8-5
8–3 ’C20x CLK Register — I/O-Space Address FFE8h 8-7
8–4 Timer Functional Block Diagram 8-8
8–5 ’C20x Timer Control Register (TCR) — I/O-Space Address FFF8h 8-11
8–6 ’C20x Wait-State Generator Control Register (WSGR) — I/O-Space Address FFFCh 8-16
8–7 $\overline{\text{BIO}}$ Timing Diagram Example 8-19
9–1 Synchronous Serial Port Block Diagram 9-3
9–2 2-Way Serial Port Transfer With External Frame Sync and External Clock 9-5
9–3 Synchronous Serial Port Control Register (SSPCR) — I/O-Space FFF1h 9-8
9–4 Burst Mode Transmission With Internal Frame Sync and Multiple Words in the Buffer 9-17
9–5 Burst Mode Transmission With External Frame Sync 9-18
9–6 Continuous Mode Transmission With Internal Frame Sync 9-20

9–7 Continuous Mode Transmission With External Frame Sync 9-21
9–8 Burst Mode Reception 9-23
9–9 Continuous Mode Reception 9-24
9–10 Test Bits in the SSPCR 9-25
9–11 Synchronous Serial Port Status (SSPST) Register — I/O address FFF2h 9-32
9–12 Synchronous Serial Port Multichannel (SSPMC) Register — FFF3h 9-34
9–13 Synchronous Serial Port Count (SSPCT) Register — FFFBh 9-38
9–14 Typical Four-Channel Codec Interface 9-41
9–15 Four-Channel 8-Bit CODEC Interface Timing Example 9-41
9–16 Four-Channel 16-Bit CODEC Interface Timing Example 9-42
10–1 Asynchronous Serial Port Block Diagram 10-3
10–2 Typical Serial Link Between a ’C20x Device and a Host CPU 10-6
10–3 Asynchronous Serial Port Control Register (ASPCR) — I/O-Space Address FFF5h 10-7
10–4 I/O Status Register (IOSR) — I/O-Space Address FFF6h 10-10
10–5 Example of the Logic for Pins IO0–IO3 10-15
10–6 Data Transmit 10-19
10–7 Data Receive 10-20
11–1 ’C209 Address Maps 11-6
11–2 ’C209 Interrupt Flag Register (IFR) — Data-Memory Address 0006h 11-12
11–3 ’C209 Interrupt Mask Register (IMR) — Data-Memory Address 0004h 11-13
11–4 ’C209 Timer Control Register (TCR) — I/O Address FFFCh 11-16
11–5 ’C209 Wait-State Generator Control Register (WSGR) — I/O Address FFFFh 11-18
B–1 ’F206 Memory Map and Serial Port Connections B-2
B–2 TMS320F206 Flash Serial Loader – ’F206 Level 1 Flow Chart B-5
D–1 Procedure for Generating Executable Files D-2
E–1 TMS320 ROM Code Submittal Flow Chart E-2
F–1 14-Pin Header Signals and Header Dimensions F-2
F–2 Emulator Cable Pod Interface F-5
F–3 Emulator Cable Pod Timings F-6
F–4 Emulator Connections Without Signal Buffering F-10
F–5 Emulator Connections With Signal Buffering F-11
F–6 Target-System-Generated Test Clock F-12
F–7 Multiprocessor Connections F-13
F–8 Pod/Connector Dimensions F-14
F–9 14-Pin Connector Dimensions F-15
F–10 Connecting a Secondary JTAG Scan Path to a Scan Path Linker F-17
F–11 EMU0/1 Configuration to Meet Timing Requirements of Less Than 25 ns F-21
F–12 Suggested Timings for the EMU0 and EMU1 Signals F-22
F–13 EMU0/1 Configuration With Additional AND Gate to Meet Timing Requirements of Greater Than 25 ns F-23
F–14 EMU0/1 Configuration Without Global Stop F-24
F–15 TBC Emulation Connections for n JTAG Scan Paths F-25

Tables

1–1 Typical Applications for TMS320 DSPs 1-3
1–2 ’C20x Generation Summary 1-4
2–1 Program and Data Memory on the TMS320C20x Devices 2-7
2–2 Serial Ports on the ’C20x Devices 2-12
3–1 Product Shift Modes for the Product-Scaling Shifter 3-7
3–2 Bit Fields of Status Registers ST0 and ST1 3-16
4–1 Pins for Interfacing With External Memory and I/O Spaces 4-3
4–2 Data Page 0 Address Map 4-8
4–3 Global Data Memory Configurations 4-11
4–4 On-Chip Registers Mapped to I/O Space 4-16
4–5 ’C203 Program-Memory Configuration Options 4-24
4–6 ’C203 Data-Memory Configuration Options 4-25
4–7 PMST Register Bit Descriptions 4-40
4–8 Bootloader-Pin Configuration 4-41
5–1 Program-Address Generation Summary 5-3
5–2 Address Loading to the Program Counter 5-4
5–3 Conditions for Conditional Branches, Calls, and Returns 5-10
5–4 Groupings of Conditions 5-11
5–5 ’C20x Interrupt Locations and Priorities 5-16
5–6 ’C20x IFR — Data-Memory Address 0006h Bit Descriptions 5-21
5–7 ’C20x IMR — Data-Memory Address 0004h Bit Descriptions 5-23
5–8 ’C20x ICR — I/O-Space Address FFECh Bit Descriptions 5-26
5–9 Reset Values of On-Chip Registers Mapped to Data Space 5-37
5–10 Reset Values of On-Chip Registers Mapped to I/O Space 5-37
5–11 Reset Conditions for the ’C206/’LC206 5-38
6–1 Indirect Addressing Operands 6-10
6–2 Effects of the ARU Code on the Current Auxiliary Register 6-13
6–3 Field Bits and Notation for Indirect Addressing 6-14
7–1 Accumulator, Arithmetic, and Logic Instructions 7-4
7–2 Auxiliary Register Instructions 7-7
7–3 TREG, PREG, and Multiply Instructions 7-8
7–4 Branch Instructions 7-9
7–5 Control Instructions 7-9
7–6 I/O and Memory Instructions 7-11
7–7 Product Shift Modes 7-37
7–8 Product Shift Modes 7-167

8–1 Peripheral Register Locations and Reset Conditions 8-2
8–2 ’C20x Input Clock Modes 8-6
8–3 ’C20x TCR — I/O Space Address FFF8h Bit Descriptions 8-11
8–4 ’C20x WSGR — I/O Space Address FFFCh Bit Descriptions 8-16
8–5 Setting the Number of Wait States With the ’C20x WSGR Bits 8-17
9–1 SSP Interface Pins 9-4
9–2 SSPCR — I/O-Space Address FFF1h Bit Descriptions 9-9
9–3 Selecting Transmit Clock and Frame Sync Sources 9-13
9–4 Run and Emulation Modes 9-26
9–5 TMS320C20x Enhanced Synchronous Serial Port Interface Signals 9-30
9–6 ESSP Registers 9-32
9–7 SSPST Register — I/O address FFF2h Bit Descriptions 9-33
9–8 SSPMC Register — FFF3h Bit Descriptions 9-35
9–9 Typical CLKX/FSX Rates and Their Prescaler Values 9-38
9–10 Options/Functions for Burst Mode and Continuous Mode 9-43
9–11 Serial Port Configuration – Burst Mode 9-44
9–12 Serial Port Configuration – Continuous Mode 9-45
10–1 Asynchronous Serial Port Interface Pins 10-4
10–2 ASPCR — I/O Space Address FFF5h Bit Descriptions 10-7
10–3 IOSR — I/O Space Address FFF6h Bit Descriptions 10-10
10–4 Common Baud Rates and the Corresponding BRD Values 10-14
10–5 Configuring Pins IO0–IO3 with ASPCR Bits CIO0–CIO3 10-16
10–6 Viewing the Status of Pins IO0–IO3 With IOSR Bits IO0–IO3 and DIO0–DIO3 10-17
11–1 ’C209 Program-Memory Configuration Options 11-8
11–2 ’C209 Data-Memory Configuration Options 11-9
11–3 ’C209 On-Chip Registers Mapped to I/O Space 11-9
11–4 ’C209 Interrupt Locations and Priorities 11-10
11–5 ’C209 IFR — Data Memory Address 0006h Bit Descriptions 11-12
11–6 ’C209 IMR — Data Memory Address 0004h Bit Descriptions 11-13
11–7 ’C209 Input Clock Modes 11-16
11–8 ’C209 TCR — I/O Address FFFCh Bit Descriptions 11-16
11–9 ’C209 WSGR — I/O Address FFFFh Bit Descriptions 11-18
A–1 Reset Values of the Status Registers A-2
A–2 Addresses and Reset Values of On-Chip Registers Mapped to Data Space A-2
A–3 Addresses and Reset Values of On-Chip Registers Mapped to I/O Space A-2
C–1 Symbols and Acronyms Used in the Instruction Set Comparison Table C-3
C–2 Summary of Enhanced Instructions C-5
D–1 Shared Programs in This Appendix D-3
D–2 Task-Specific Programs in This Appendix D-3
F–1 14-Pin Header Signal Descriptions F-3
F–2 Emulator Cable Pod Timing Parameters F-6

Examples

4–1 An Interrupt Service Routine Supporting $\overline{INT1}$ and $\overline{HOLD}$ 4-19
6–1 RPT Instruction Using Short-Immediate Addressing 6-2
6–2 ADD Instruction Using Long-Immediate Addressing 6-3
6–3 Using Direct Addressing with ADD (Shift of 0 to 15) 6-7
6–4 Using Direct Addressing with ADD (Shift of 16) 6-7
6–5 Using Direct Addressing with ADDC 6-8
6–6 Selecting a New Current Auxiliary Register 6-12
6–7 No Increment or Decrement 6-15
6–8 Increment by 1 6-15
6–9 Decrement by 1 6-16
6–10 Increment by Index Amount 6-16
6–11 Decrement by Index Amount 6-16
6–12 Increment by Index Amount With Reverse Carry Propagation 6-16
6–13 Decrement by Index Amount With Reverse Carry Propagation 6-16
D–1 Generic Command File (c203.cmd) D-5
D–2 Header File With I/O Register Declarations (init.h) D-6
D–3 Header File With Interrupt Vector Declarations (vector.h) D-7
D–4 Implementing Simple Delay Loops (delay.asm) D-8
D–5 Testing and Using the Timer (timer.asm) D-9
D–6 Testing and Using Interrupt $\overline{INT1}$ (intr1.asm) D-10
D–7 Implementing a HOLD Operation (hold.asm) D-11
D–8 Testing and Using Interrupts $\overline{INT2}$ and $\overline{INT3}$ (intr23.asm) D-12
D–9 Asynchronous Serial Port Transmission (uart.asm) D-13
D–10 Loopback to Verify Transmissions of Asynchronous Serial Port (echo.asm) D-14
D–11 Testing and Using Automatic Baud-Rate Detection on Asynchronous Serial Port (autobaud.asm) D-16
D–12 Testing and Using Asynchronous Serial Port Delta Interrupts (bitio.asm) D-18
D–13 Synchronous Serial Port Continuous Mode Transmission (ssp.asm) D-20
D–14 Using Synchronous Serial Port With Codec Device (ad55.asm) D-21
D–15 Linker Command File D-24
D–16 Hex Conversion Utility Command File D-24
F–1 Key Timing for a Single-Processor System Without Buffers F-8
F–2 Key Timing for a Single- or Multiple-Processor System With Buffered Input and Output F-8
F–3 Key Timing for a Single-Processor System Without Buffering (SPL) F-19
F–4 Key Timing for a Single- or Multiprocessor-System With Buffered Input and Output (SPL) F-19

Chapter 1

Introduction

The TMS320C20x ('C20x) is one of several fixed-point generations of DSPs in the TMS320 family. The 'C20x is source-code compatible with the TMS320C2x. Much of the code written for the 'C2x can be reassembled to run on a 'C20x device. In addition, the 'C20x generation is upward compatible with the 'C5x generation of DSPs.

Topic **Page**

1.1 TMS320 Family **1-2**

1.2 TMS320C20x Generation **1-4**

1.3 Key Features of the TMS320C20x **1-5**

1.1 TMS320 Family

The TMS320 family consists of fixed-point, floating-point, and multiprocessor digital signal processors (DSPs). TMS320 DSPs have an architecture designed specifically for real-time signal processing. The following characteristics make this family the ideal choice for a wide range of processing applications:

- ❑ Flexible instruction sets
- ❑ High-speed performance
- ❑ Innovative parallel architectures
- ❑ 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, *Electronic Products* magazine awarded the TMS32010 the “Product of the Year” title. The next generation devices continue meeting new performance levels for TI DSPs.

Devices within a generation of the TMS320 family have the same CPU structure but different on-chip memory and peripheral configurations. Spin-off devices use new combinations of on-chip memory and peripherals to satisfy a wide range of needs in the worldwide electronics market. By integrating memory and peripherals onto a single chip, TMS320 devices reduce system cost and save circuit board space.

1.1.2 Typical Applications for the TMS320 Family

Table 1–1 lists some typical applications for the TMS320 family of DSPs. The TMS320 DSPs offer adaptable approaches to traditional signal-processing problems such as filtering and vocoding. They also support complex applications that often require multiple operations to be performed simultaneously.

Table 1–1. Typical Applications for TMS320 DSPs

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	Pagers	Motor control
Engine control	Power tools	Robotics control
Global positioning	Radar detectors	Servo control
Navigation	Solid-state answering machines	
Vibration analysis		
Voice commands		
General-Purpose	**Graphics/Imaging**	**Industrial**
Adaptive filtering	3-D rotation	Numeric control
Convolution	Animation/digital maps	Power-line monitoring
Correlation	Homomorphic processing	Robotics
Digital filtering	Image compression/transmission	Security access
Fast Fourier transforms	Image enhancement	
Hilbert transforms	Pattern recognition	
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 28 800-bps modems	Faxing	Speaker verification
Adaptive equalizers	Line repeaters	Speech enhancement
ADPCM transcoders	Personal communications systems (PCS)	Speech recognition
Cellular telephones	Personal digital assistants (PDA)	Speech synthesis
Channel multiplexing	Speaker phones	Speech vocoding
Data encryption	Spread spectrum communications	Text-to-speech applications
Digital PBXs	Video conferencing	Voice mail
Digital speech interpolation (DSI)	X.25 packet switching	
DTMF encoding/decoding		
Echo cancellation		

1.2 TMS320C20x Generation

Texas Instruments uses static CMOS integrated-circuit technology to fabricate the TMS320C20x DSPs. The architectural design of the 'C20x is based on that of the 'C5x. The operational flexibility and speed of the 'C20x and 'C5x are a result of an advanced, modified Harvard architecture (which has separate buses for program and data memory), a multilevel pipeline, on-chip peripherals, on-chip memory, and a highly specialized instruction set. The 'C20x performs up to 40 MIPS (million instructions per second).

The 'C20x generation offers the following benefits:

- ❑ Enhanced TMS320 architectural design for increased performance and versatility
- ❑ Modular architectural design for fast development of additional spin-off devices
- ❑ Advanced IC processing technology for increased performance
- ❑ Fast and easy performance upgrades for 'C1x and 'C2x source code, which is upward compatible with 'C20x source code
- ❑ Enhanced instruction set for faster algorithms and for optimized high-level language operation
- ❑ New static design techniques for minimizing power consumption

Table 1–2 provides an overview of the basic features of the 'C20x DSPs.

Table 1–2. 'C20x Generation Summary

Device	Cycle Time (ns)	Operating Voltage (Vdd)	On-Chip Memory			MEM	Serial Ports		I/O		Timers	Package
			RAM	ROM	Flash	Off-Chip	Sync	Async	PAR	DMA		
TMS320C203	25/35/50	5V	544	–	–	192K	1	1	64K x 16	Ext.	1	100 TQFP†
TMS320LC203	50	3.3V	544	–	–	192K	1	1	64K x 16	Ext.	1	100 TQFP†
TMS320F206	50	5V	4.5K	–	32K	192K	1	1	64K x 16	Ext.	1	100 TQFP†
TMS320C209	35/50	5V	4.5K	4K	–	192K	–	–	64K x 16	Ext.	1	80 TQFP†
TMS320C206	25	3.3V	4.5K	32K	–	192K	1	1	64K x 16	Ext.	1	100 TQFP†
TMS320LC206	25	3.3V	4.5K	32K	–	192K	1	1	64K x 16	Ext.	1	100 TQFP†

† TQFP = Thin quad flat pack

1.3 Key Features of the TMS320C20x

Key features on the various ’C20x devices are:

- Speed:
  - 50-, 35-, or 25-ns execution time of a single-cycle instruction
  - 20, 28.5, or 40 MIPS
- Code compatibility with other TMS320 fixed-point devices:
  - Source-code compatible with all ’C1x and ’C2x devices
  - Upward compatible with the ’C5x devices
- Memory:
  - 224K words of addressable memory space (64K words of program space, 64K words of data space, 64K words of I/O space, and 32K words of global space)
  - 544 words of dual-access on-chip RAM (288 words for data and 256 words for program/data)
  - 32K words on-chip ROM or 32K words on-chip flash memory (on ’C206 and ’F206)
  - 4K words of single-access on-chip RAM (on ’C206 and ’F206)
- CPU:
  - 32-bit arithmetic logic unit (CALU)
  - 32-bit accumulator
  - 16-bit $\times$ 16-bit parallel multiplier with 32-bit product capability
  - Three scaling shifters
  - Eight 16-bit auxiliary registers with a dedicated arithmetic unit for indirect addressing of data memory
- Program control:
  - 4-level pipeline operation
  - 8-level hardware stack
  - User-maskable interrupt lines

- Instruction set:
  - Single-instruction repeat operation
  - Single-cycle multiply/accumulate instructions
  - Memory block move instructions for better program/data management
  - Indexed-addressing capability
  - Bit-reversed indexed-addressing capability for radix-2 FFTs
- On-chip peripherals:
  - Software-programmable timer
  - Software-programmable wait-state generator for program, data, and I/O memory spaces
  - Oscillator and phase-locked loop (PLL) to implement clock options: ×1, ×2, ×4, and ÷2 (only ×2 and ÷2 available on ’C209)
  - CLK register for turning the CLKOUT1 pin on and off (not available on ’C209)
  - Synchronous serial port (not available on ’C209)
  - Asynchronous serial port (not available on ’C209)
- On-chip scanning-logic circuitry (IEEE Standard 1149.1) for emulation and testing purposes
- Power:
  - 5- or 3.3-V static CMOS technology
  - Power-down mode to reduce power consumption
- Packages:
  - 100-pin TQFP (thin quad flat pack)
  - 80-pin TQFP for the ’C209

Chapter 2

Architectural Overview

This chapter provides an overview of the architectural structure and components of the ’C20x. The ’C20x DSPs use an advanced, modified Harvard architecture that maximizes processing power by maintaining separate bus structures for program memory and data memory. The three main components of the ’C20x are the central processing unit (CPU), memory, and on-chip peripherals.

Figure 2–1 shows an overall block diagram of the ’C20x.

Note:

All ’C20x devices use the same central processing unit (CPU), bus structure, and instruction set, but the ’C209 has some notable differences. For example, although certain peripheral control registers have the same names on all ’C20x devices, these registers are located at different I/O addresses on the ’C209. See Chapter 11 for a detailed description of the differences on the ’C209.

Topic **Page**

2.1 ’C20x Bus Structure **2-3**

2.2 Central Processing Unit **2-5**

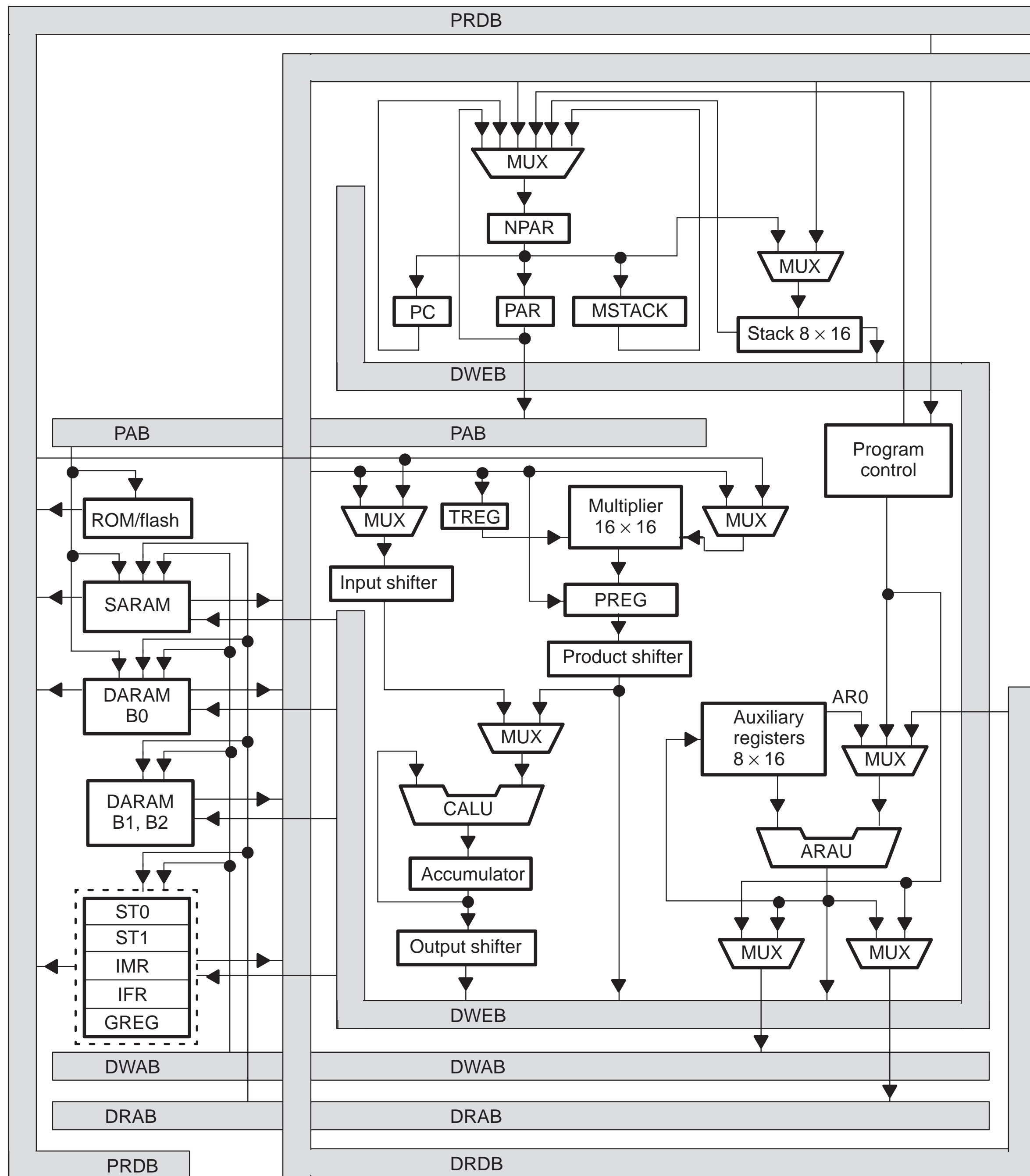
2.3 Memory and I/O Spaces **2-7**

2.4 Program Control **2-10**

2.5 On-Chip Peripherals **2-11**

2.6 Scanning-Logic Circuitry **2-13**

Figure 2–1. Overall Block Diagram of the ’C20x



Note: The I/O-mapped (peripheral) registers are not part of the core; they are accessed as shown in Figure 2–2 on page 2-4.

2.1 ’C20x Bus Structure

Figure 2–2 shows a block diagram of the ’C20x bus structure. The ’C20x internal architecture is built around six 16-bit buses:

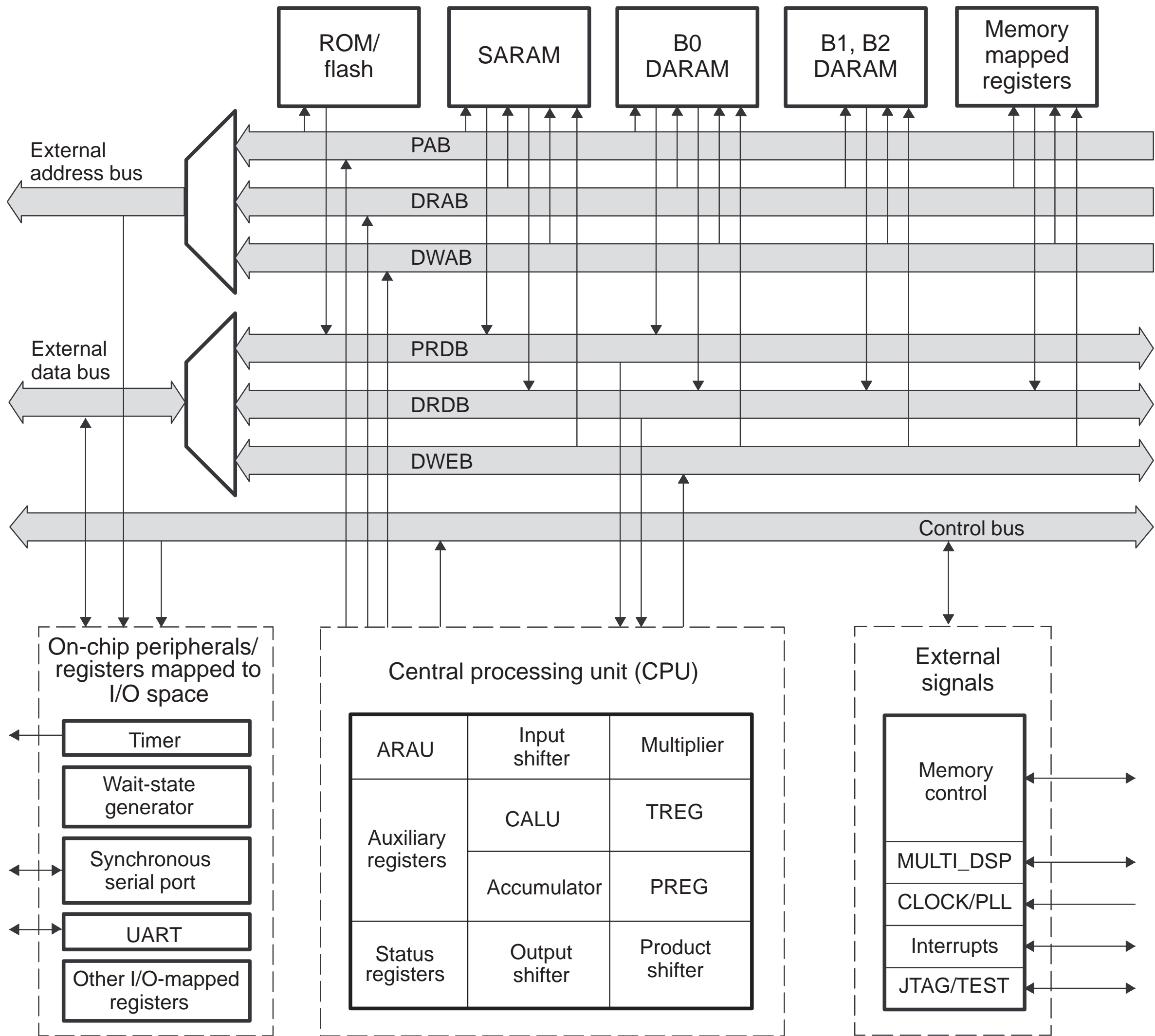
- ❑ PAB. The *program address bus* provides addresses for both reads from and writes to program memory.
- ❑ DRAB. The *data-read address bus* provides addresses for reads from data memory.
- ❑ DWAB. The *data-write address bus* provides addresses for writes to data memory.
- ❑ PRDB. The *program read bus* carries instruction code and immediate operands, as well as table information, from program memory to the CPU.
- ❑ DRDB. The *data read bus* carries data from data memory to the central arithmetic logic unit (CALU) and the auxiliary register arithmetic unit (ARAU).
- ❑ DWEB. The *data write bus* carries data to both program memory and data memory.

Having separate address buses for data reads (DRAB) and data writes (DWAB) allows the CPU to read and write in the same machine cycle.

Separate program and data spaces allow simultaneous access to program instructions and data. For example, while data is multiplied, a previous product can be added to the accumulator, and, at the same time, a new address can be generated. Such parallelism supports a set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. In addition, the ’C20x includes control mechanisms to manage interrupts, repeated operations, and function/subroutine calls.

All ’C20x devices share the same CPU and bus structure; however, each device has different on-chip memory configurations and on-chip peripherals.

Figure 2–2. Bus Structure Block Diagram



2.2 Central Processing Unit

The CPU is the same on all the ’C20x devices. The ’C20x CPU contains:

- A 32-bit central arithmetic logic unit (CALU)
- A 32-bit accumulator
- Input and output data-scaling shifters for the CALU
- A 16-bit × 16-bit multiplier
- A product-scaling shifter
- Data-address generation logic, which includes eight auxiliary registers and an auxiliary register arithmetic unit (ARAU)
- Program-address generation logic

2.2.1 Central Arithmetic Logic Unit (CALU) and Accumulator

The ’C20x performs 2s-complement arithmetic using the 32-bit CALU. The CALU uses 16-bit words taken from data memory or derived from an immediate instruction, or it uses the 32-bit result from the multiplier. In addition to arithmetic operations, the CALU can perform Boolean operations.

The accumulator stores the output from the CALU; it can also provide a second input to the CALU. The accumulator is 32 bits wide and is divided into a high-order word (bits 31 through 16) and a low-order word (bits 15 through 0). Assembly language instructions are provided for storing the high- and low-order accumulator words to data memory.

2.2.2 Scaling Shifters

The ’C20x has three 32-bit shifters that allow for scaling, bit extraction, extended arithmetic, and overflow-prevention operations:

- Input data-scaling shifter (input shifter). This shifter left shifts 16-bit input data by 0 to 16 bits to align the data to the 32-bit input of the CALU.

- Output data-scaling shifter (output shifter). This shifter can left shift output from the accumulator by 0 to 7 bits before the output is stored to data memory. The content of the accumulator remains unchanged.

- Product-scaling shifter (product shifter). The product register (PREG) receives the output of the multiplier. The product shifter shifts the output of the PREG before that output is sent to the input of the CALU. The product shifter has four product shift modes (no shift, left shift by one bit, left shift by four bits, and right shift by 6 bits), which are useful for performing multiply/accumulate operations, performing fractional arithmetic, or justifying fractional products.

2.2.3 Multiplier

The on-chip multiplier performs 16-bit × 16-bit 2s-complement multiplication with a 32-bit result. In conjunction with the multiplier, the ’C20x uses the 16-bit temporary register (TREG) and the 32-bit product register (PREG). The TREG always supplies one of the values to be multiplied. The PREG receives the result of each multiplication.

Using the multiplier, TREG, and PREG, the ’C20x efficiently performs fundamental DSP operations such as convolution, correlation, and filtering. The effective execution time of each multiplication instruction can be as short as one CPU cycle.

2.2.4 Auxiliary Register Arithmetic Unit (ARAU) and Auxiliary Registers

The ARAU generates data memory addresses when an instruction uses indirect addressing (see Chapter 6, *Addressing Modes*) to access data memory. The ARAU is supported by eight auxiliary registers (AR0 through AR7), each of which can be loaded with a 16-bit value from data memory or directly from an instruction word. Each auxiliary register value can also be stored to data memory. The auxiliary registers are referenced by a 3-bit auxiliary register pointer (ARP) embedded in status register ST0.

2.3 Memory and I/O Spaces

The ’C20x memory is organized into four individually selectable spaces: program, local data, global data, and I/O. These spaces form an address range of 224K words.

All ’C20x devices include 288 words of dual-access RAM (DARAM) for data memory and 256 words of data/program DARAM. Depending on the device, it may also have data/program single-access RAM (SARAM) and read-only memory (ROM) or flash memory. Table 2–1 shows how much ROM, flash memory, DARAM, and SARAM are available on the different ’C20x devices.

Table 2–1. Program and Data Memory on the TMS320C20x Devices

Memory Type	**’C203**	**’C206†**	**’F206**	**’C209**
ROM (words)	–	32K	–	4K
Flash memory (words)	–	–	32K	–
DARAM (words)	544	544	544	544
Data (words)	288	288	288	288
Data/program (words)	256	256	256	256
SARAM (words)	–	4K	4K	4K

† ’C206 refers to the ’C206/’LC206 unless specified otherwise.

The ’C20x also has CPU registers that are mapped in data memory space and peripheral registers that are mapped in on-chip I/O space. The ’C20x memory types and features are introduced in the sections following this paragraph. For more details about the configuration and use of the ’C20x memory and I/O space, see Chapter 4, *Memory and I/O Space*.

2.3.1 Dual-Access On-Chip RAM

All ’C20x devices have 544 words × 16-bits of on-chip DARAM, which can be accessed twice per machine cycle. This memory is primarily intended to hold data but, when needed, can also hold programs. It can be configured in one of two ways:

- All 544 words are configured as data memory.
- 288 words are configured as data memory, and 256 words are configured as program memory.

Because DARAM can be accessed twice per cycle, it improves the speed of the CPU. The CPU operates within a four-cycle pipeline. In this pipeline, the

CPU reads data on the third cycle and writes data on the fourth cycle. However, DARAM allows the CPU to write and read in one cycle; the CPU writes to DARAM on the master phase of the cycle and reads from DARAM on the slave phase. For example, suppose two instructions, A and B, store the accumulator value to DARAM and load the accumulator with a new value from DARAM. Instruction A stores the accumulator value during the master phase of the CPU cycle, and instruction B loads the new value to the accumulator during the slave phase. Because part of the dual-access operation is a write, it only applies to RAM.

2.3.2 Single-Access On-Chip Program/Data RAM

Some of the ’C20x devices have 4K 16-bit words of single-access RAM (SARAM). The addresses associated with the SARAM can be used for both data memory and program memory and are software- or hardware-configurable (depending on the device) to either external memory or the internal SARAM. When configured as external, these addresses can be used for off-chip data and program memory. Code can be booted from off-chip ROM and then executed at full speed once it is loaded into the on-chip SARAM. Because the SARAM can be mapped to program and/or data memory, the SARAM allows for more flexible address mapping than the DARAM block.

SARAM is accessed only once per CPU cycle. When the CPU requests multiple accesses, the SARAM schedules the accesses by providing a not-ready condition to the CPU and then executing the accesses one per cycle. For example, if the instruction sequence involves storing the accumulator value and then loading a value to the accumulator, it would take two cycles to complete in SARAM, compared to one cycle in DARAM.

2.3.3 Factory-Masked On-Chip ROM

’C206/’LC206 devices feature an on-chip, 32K 16-bit words of programmable ROM. The ROM can be selected during reset by driving the MP/$\overline{\text{MC}}$ pin low. If the ROM is not selected, the device starts its execution from off-chip memory.

If you want a custom ROM, you can provide the code or data to be programmed into the ROM in object file format, and Texas Instruments will generate the appropriate process mask to program the ROM. See Appendix E for details on how to submit ROM code to Texas Instruments.

2.3.4 Flash Memory

Some of the ’C20x devices feature on-chip blocks of flash memory, which is electronically erasable and programmable, and non-volatile. Each block of flash memory will have a set of control registers that allow for erasing, programming, and testing of that block. The flash memory blocks can be selected during reset by driving the MP/$\overline{\text{MC}}$ pin low. If the flash memory is not selected, the device starts its execution from off-chip memory. For a further description on the TMS320F2xx flash devices and how they are used, please refer to the flash technical reference, *TMS320F2xx Flash Memory Technical Reference* (literature number SPRU282).

2.4 Program Control

Several features provide program control:

- The program controller of the CPU decodes instructions, manages the pipeline, stores the status of operations, and decodes conditional operations. Elements involved in program control are the program counter, the status registers, the stack, and the address-generation logic.
- Software mechanisms used for program control include branches, calls, conditional instructions, a repeat instruction, reset, and interrupts.

For descriptions of these program control features, see Chapter 5, *Program Control.*

2.5 On-Chip Peripherals

All the ’C20x devices have the same CPU, but different on-chip peripherals are connected to their CPUs. The on-chip peripherals featured on the ’C20x devices are:

- ❑ Clock generator (an oscillator and a phase lock loop circuit)
- ❑ CLK register for turning the CLKOUT1 pin on and off
- ❑ Timer
- ❑ Wait-state generator
- ❑ General-purpose input/output (I/O) pins
- ❑ Synchronous serial port
- ❑ Asynchronous serial port

2.5.1 Clock Generator

The clock generator consists of an internal oscillator and an internal phase lock loop (PLL) circuit. The clock generator can be driven internally by connecting the DSP to a crystal resonator circuit, or it can be driven by an external clock source. The PLL circuit generates an internal CPU clock by multiplying the clock source by a specified factor. Thus, you can use a clock source with a lower frequency than that of the CPU. The clock generator is discussed in section 8.2, on page 8-4.

2.5.2 CLKOUT1-Pin Control (CLK) Register

The ’C20x CLK register controls whether the master clock output signal (CLKOUT1) is available at the CLKOUT1 pin.

2.5.3 Hardware Timer

The ’C20x features a 16-bit down-counting timer with a 4-bit prescaler. Timer control bits can stop, start, reload, and determine the prescaler count for the timer. For more information, see section 8.4, *Timer*, on page 8-8.

2.5.4 Software-Programmable Wait-State Generator

Software-programmable wait-state logic is incorporated (without any external hardware) for interfacing with slower off-chip memory and I/O devices. The ’C209 wait-state generator generates zero or one wait states; the wait-state generator on other ’C20x devices generates zero to seven wait states. For more information, see section 8.5, *Wait-State Generator*, on page 8-15.

2.5.5 General-Purpose I/O Pins

The ’C20x has pins that provide general-purpose input or output signals. All ’C20x devices have a general-purpose input pin, $\overline{BIO}$, and a general-purpose output pin, XF. Except for the ’C209, the ’C20x devices also have pins IO0, IO1, IO2, and IO3, which are connected to corresponding bits (IO0–IO3) mapped into the on-chip I/O space. These bits can be individually configured as inputs or outputs. For more information on the general-purpose pins, see section 8.6, on page 8-18.

2.5.6 Serial Ports

The serial ports available on the ’C20x vary by device, but two types of serial ports are represented: synchronous and asynchronous. See Table 2–2 for the number of each kind on the various ’C20x devices. The sections following the table provide an introduction to the two types of serial ports.

Table 2–2. Serial Ports on the ’C20x Devices

Serial Ports	’C203	’C206	’F206	’C209
Synchronous	1	1	1	–
Asynchronous	1	1	1	–

Synchronous serial port (SSP)

The ’C20x synchronous serial port (SSP) communicates with codecs, other ’C20x devices, and external peripherals. The SSP offers:

- Two four-word-deep first in, first out (FIFO) buffers that have interrupt-generating capabilities.
- Burst and continuous transfer modes.
- A wide range of operation speeds when external clocking is used.

If internal clocking is used, the speed is fixed at 1/2 of the internal DSP clock frequency. For more information on the SSP, see Chapter 9.

Asynchronous serial port (ASP)

The ’C20x asynchronous serial port (ASP) communicates with asynchronous serial devices. The ASP has a maximum transfer rate of 250,000 characters per second (assuming it uses10 bits to transmit each 8-bit character). The ASP also has logic for automatic baud detection, which allows the ASP to lock to the incoming data rate. All transfers through the asynchronous serial port use double buffering. See Chapter 10, *Asynchronous Serial Port,* for more information.

2.6 Scanning-Logic Circuitry

The ’C20x has JTAG scanning-logic circuitry that is compatible with IEEE Standard 1149.1. This circuitry is used for emulation and testing purposes only. The serial scan path is used to perform operational tests on the on-chip peripherals. The internal scanning logic provides access to all of the on-chip resources. Thus, the serial-scan pins and the emulation pins on ’C20x devices allow on-board emulation. However, on all ’C20x devices, the serial scan path does not have boundary scan logic. Appendix F provides information to help you meet the design requirements of the Texas Instruments XDS510™ emulator with respect to IEEE-1149.1 designs and discusses the XDS510 cable.

Chapter 3

Central Processing Unit

This chapter describes the main components of the central processing unit (CPU). First, this chapter describes three fundamental sections of the CPU (see Figure 3–1):

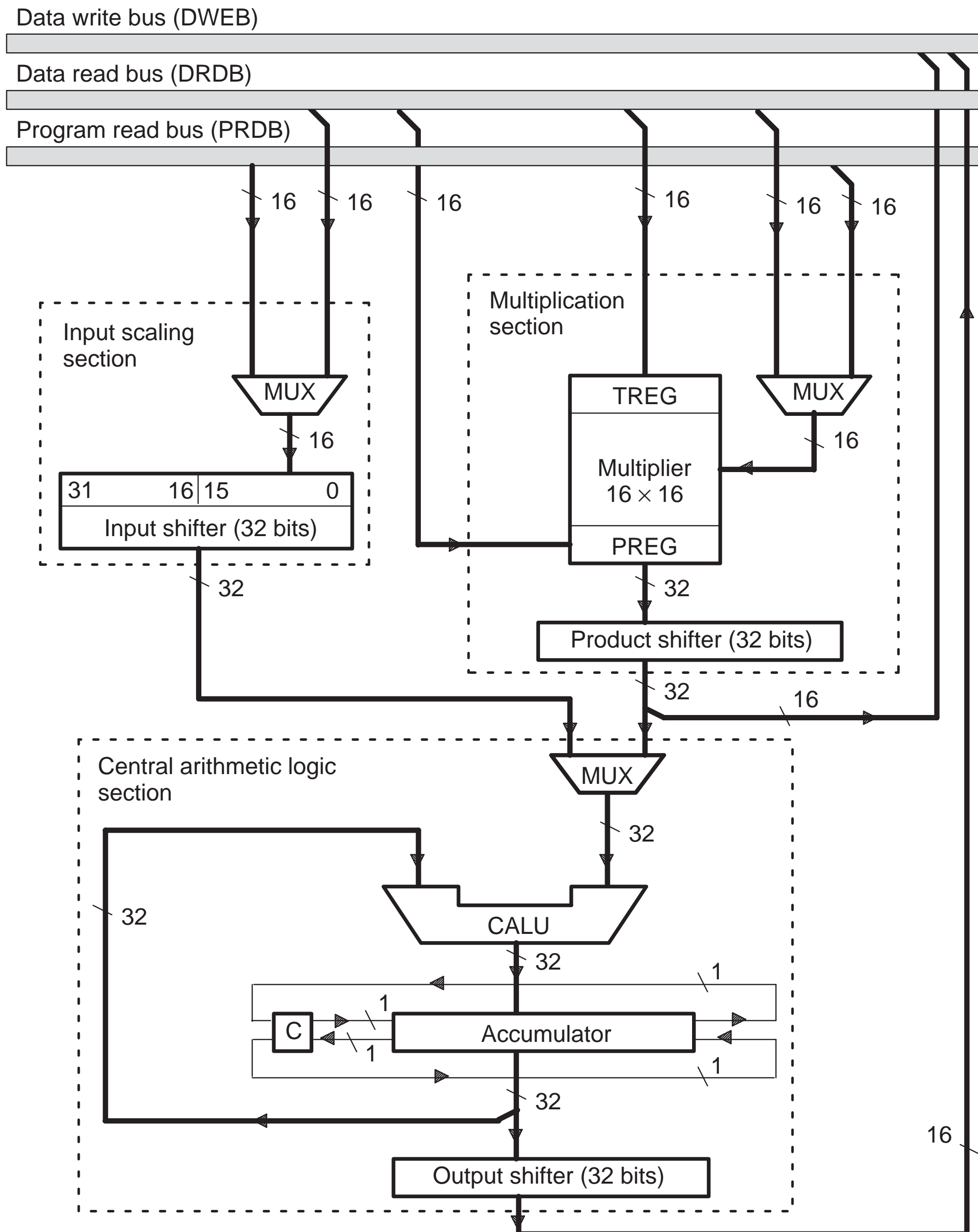
- Input scaling section
- Multiplication section
- Central arithmetic logic section

The chapter then describes the auxiliary register arithmetic unit (ARAU), which performs arithmetic operations independently of the central arithmetic logic section. The chapter concludes with a description of status registers ST0 and ST1, which contain bits for determining processor modes, addressing pointer values, and indicating various processor conditions and arithmetic logic results.

Topic **Page**

3.1 Input Scaling Section 3-3
3.2 Multiplication Section 3-5
3.3 Central Arithmetic Logic Section 3-8
3.4 Auxiliary Register Arithmetic Unit (ARAU) 3-12
3.5 Status Registers ST0 and ST1 3-15

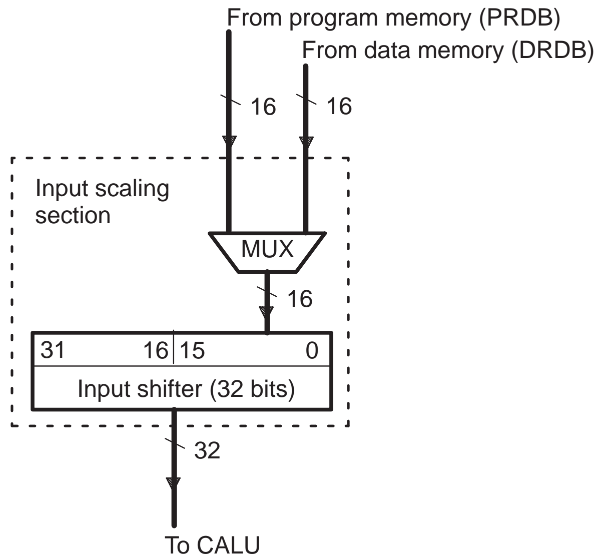
Figure 3–1. Block Diagram of the Input Scaling, Central Arithmetic Logic, and Multiplication Sections of the CPU



3.1 Input Scaling Section

A 32-bit input data-scaling shifter (input shifter) aligns a 16-bit value coming from memory to the 32-bit CALU. This data alignment is necessary for data-scaling arithmetic as well as aligning masks for logical operations. The input shifter operates as part of the data path between program or data space and the CALU and, thus, requires no cycle overhead. Described directly below are the input, the output, and the shift count of the input shifter. Throughout the discussion, refer to Figure 3–2.

Figure 3–2. Block Diagram of the Input Scaling Section



Input. Bits 15 through 0 of the input shifter accept a 16-bit input from either of two sources (see Figure 3–2):

- *The data read bus (DRDB).* This input is a value from a data memory location referenced in an instruction operand.
- *The program read bus (PRDB).* This input is a constant value given as an instruction operand.

Output. After a value has been accepted into bits 15 through 0, the input shifter aligns the16-bit value to the 32-bit bus of the CALU as shown in Figure 3–2. The shifter shifts the value left 0 to 16 bits and then sends the 32-bit result to the CALU.

During the left shift, unused LSBs in the shifter are filled with zeros, and unused MSBs in the shifter are either filled with zeros or sign extended, depending on the value of the sign-extension mode bit (SXM) of status register ST1.

Shift count. The shifter can left-shift a 16-bit value by 0 to 16 bits. The size of the shift (or the shift count) is obtained from one of two sources:

- *A constant embedded in the instruction word.* Putting the shift count in the instruction word allows you to use specific data-scaling or alignment operations customized for your program code.
- *The four LSBs of the temporary register (TREG).* The TREG-based shift allows the data-scaling factor to be determined dynamically so that it can be adapted to the system's performance.

Sign-extension mode bit. For many but not all instructions, the sign-extension mode bit (SXM), bit 10 of status register ST1, determines whether the CALU uses sign extension during its calculations. If SXM = 0, sign extension is suppressed. If SXM = 1, the output of the input shifter is sign extended. Figure 3–3 shows an example of an input value shifted left by 8 bits for SXM = 0. The MSBs of the value passed to the CALU are zero filled. Figure 3–4 shows the same shift but with SXM = 1. The value is sign extended during the shift.

Figure 3–3. Operation of the Input Shifter for SXM = 0

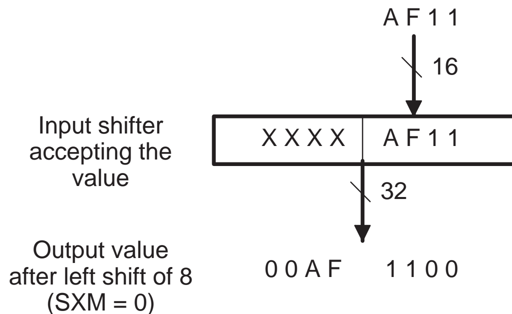
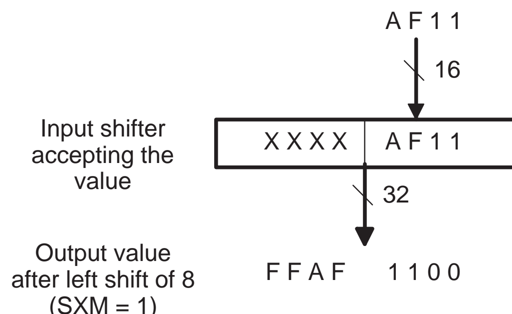


Figure 3–4. Operation of the Input Shifter for SXM = 1

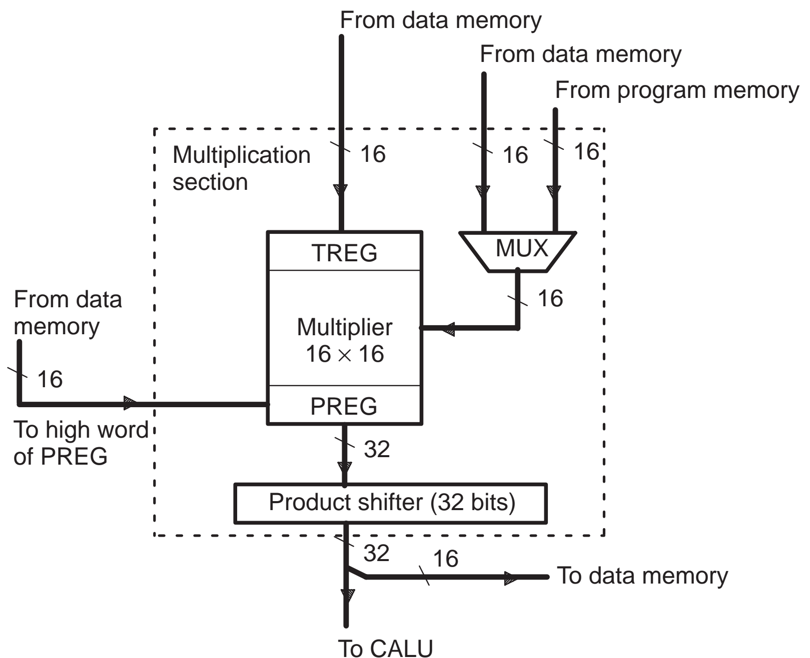


3.2 Multiplication Section

The ’C20x uses a 16-bit × 16-bit hardware multiplier that can produce a signed or unsigned 32-bit product in a single machine cycle. As shown in Figure 3–5, the multiplication section consists of:

- The 16-bit temporary register (TREG), which holds one of the multiplicands
- The multiplier, which multiplies the TREG value by a second value from data memory or program memory
- The 32-bit product register (PREG), which receives the result of the multiplication
- The product shifter, which scales the PREG value before passing it to the CALU.

Figure 3–5. Block Diagram of the Multiplication Section



3.2.1 Multiplier

The 16-bit × 16-bit hardware multiplier can produce a signed or unsigned 32-bit product in a single machine cycle. The two numbers being multiplied are treated as 2s-complement numbers, except during unsigned multiplication (MPYU instruction). Descriptions of the inputs and output of the multiplier follow.

Inputs. The multiplier accepts two 16-bit inputs:

- One input is always from the 16-bit temporary register (TREG). The TREG is loaded before the multiplication with a data-value from the data read bus (DRDB).
- The other input is one of the following:
  - A data-memory value from the data read bus (DRDB).
  - A program memory value from the program read bus (PRDB).

Output. After the two 16-bit inputs are multiplied, the 32-bit result is stored in the product register (PREG). The output of the PREG is connected to the 32-bit product-scaling shifter. Through this shifter, the product may be transferred from the PREG to the CALU or to data memory (by the SPH and SPL instructions).

3.2.2 Product-Scaling Shifter

The product-scaling shifter (product shifter) facilitates scaling of the product register (PREG) value. The shifter has a 32-bit input connected to the output of the PREG and a 32-bit output connected to the input of the CALU.

Input. The shifter has a 32-bit input connected to the output of the PREG.

Output. After the shifter completes the shift, all 32 bits of the result can be passed to the CALU, or 16 bits of the result can be stored to data memory.

Shift Modes. This shifter uses one of four product shift modes, summarized in Table 3–1. As shown in the table, these modes are determined by the product shift mode (PM) bits of status register ST1. In the first shift mode (PM = 00), the shifter does not shift the product at all before giving it to the CALU or to data memory. The next two modes cause left shifts (of one or four), which are useful for implementing fractional arithmetic or justifying products. The right-shift mode shifts the product by six bits, enabling the execution of up to 128 consecutive multiply-and-accumulate operations without causing the accumulator to overflow. Note that the content of the PREG remains unchanged; the value is copied to the product shifter and shifted there.

Note:

The right shift in the product shifter is always sign extended, regardless of the value of the sign-extension mode bit (SXM) of status register ST1.

Table 3–1. Product Shift Modes for the Product-Scaling Shifter

PM	Shift	Comments
00	No shift	Product sent to CALU or data write bus (DWEB) with no shift
01	Left by 1 shift	Removes the extra sign bit generated in a 2s-complement multiply to produce a Q31 product†
10	Left by 4 bits	Removes the extra four sign bits generated in a 16-bit × 13-bit 2s-complement multiply to produce a Q31 product† when multiplying by a 13-bit constant
11	Right by 6 bits	Scales the product to allow up to 128 product accumulations without overflowing the accumulator. The right shift is always sign extended, regardless of the value of the sign-extension mode bit (SXM) of status register ST1.

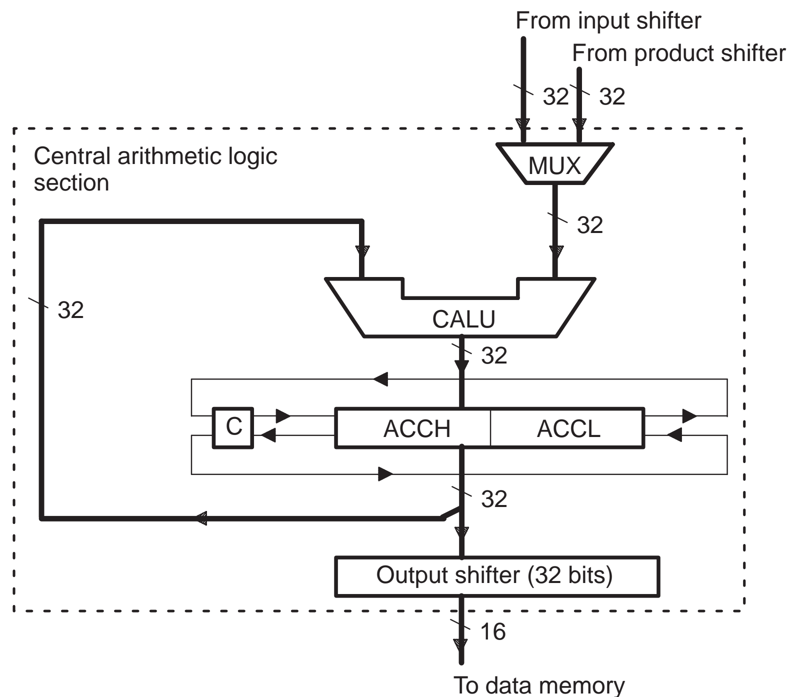
† A Q31 number is a binary fraction in which there are 31 digits to the right of the binary point (the base 2 equivalent of the base 10 decimal point).

3.3 Central Arithmetic Logic Section

Figure 3–6 shows the main components of the central arithmetic logic section, which are:

- ❑ The central arithmetic logic unit (CALU), which implements a wide range of arithmetic and logic functions.
- ❑ The 32-bit accumulator (ACC), which receives the output of the CALU and is capable of performing bit shifts on its contents with the help of the carry bit (C). Figure 3–6 shows the accumulator's high word (ACCH) and low word (ACCL).
- ❑ The output shifter, which can shift a copy of either the high word or low word of the accumulator before sending it to data memory for storage.

Figure 3–6. Block Diagram of the Central Arithmetic Logic Section



3.3.1 Central Arithmetic Logic Unit (CALU)

The central arithmetic logic unit (CALU), implements a wide range of arithmetic and logic functions, most of which execute in a single clock cycle. These functions can be grouped into four categories:

- 16-bit addition
- 16-bit subtraction
- Boolean logic operations
- Bit testing, shifting, and rotating.

Because the CALU can perform Boolean operations, you can perform bit manipulation. For bit shifting and rotating, the CALU uses the accumulator. The CALU is referred to as central because there is an independent arithmetic unit, the auxiliary register arithmetic unit (ARAU), which is described in section 3.4. A description of the inputs, the output, and an associated status bit of the CALU follows.

Inputs. The CALU has two inputs (see Figure 3–6):

- One input is always provided by the 32-bit accumulator.
- The other input is provided by one of the following:
  - The product-scaling shifter (see section 3.2.2)
  - The input data-scaling shifter (see section 3.1)

Output. Once the CALU performs an operation, it transfers the result to the 32-bit accumulator, which is capable of performing bit shifts of its contents. The output of the accumulator is connected to the 32-bit output data-scaling shifter. Through the output shifter, the accumulator's upper and lower 16-bit words can be individually shifted and stored to data memory.

Sign-extension mode bit. For many but not all instructions, the sign-extension mode bit (SXM), bit 10 of status register ST1, determines whether the CALU uses sign extension during its calculations. If SXM = 0, sign extension is suppressed. If SXM = 1, sign extension is enabled.

3.3.2 Accumulator

Once the CALU performs an operation, it transfers the result to the 32-bit accumulator, which can then perform single-bit shifts or rotations on its contents. Each of the accumulator's upper and lower 16-bit words can be passed to the output data-scaling shifter, where it can be shifted, and then stored in data memory. Status bits and branch instructions associated with the accumulator are discussed directly below.

Status bits. Four status bits are associated with the accumulator:

- *Carry bit (C).* C (bit 9 of status register ST1) is affected during:
  - Additions to and subtractions from the accumulator:

C = 0	When the result of a subtraction generates a borrow.
	When the result of an addition does not generate a carry. (Exception: When the ADD instruction is used with a shift of 16 and no carry is generated, the ADD instruction has no affect on C.)
C = 1	When the result of an addition generates a carry.
	When the result of a subtraction does not generate a borrow. (Exception: When the SUB instruction is used with a shift of 16 and no borrow is generated, the SUB instruction has no effect on C.)

  - Single-bit shifts and rotations of the accumulator value. During a left shift or rotation, the most significant bit of the accumulator is passed to C; during a right shift or rotation, the least significant bit is passed to C.

- *Overflow mode bit (OVM).* OVM (bit 11 of status register ST0) determines how the accumulator will reflect arithmetic overflows. When the processor is in overflow mode (OVM = 1) and an overflow occurs, the accumulator is filled with one of two specific values:
  - If the overflow is in the positive direction, the accumulator is filled with its most positive value (7FFF FFFFh).
  - If the overflow is in the negative direction, the accumulator is filled with its most negative value (8000 0000h).

- *Overflow flag bit (OV).* OV is bit 12 of status register ST0. When no accumulator overflow is detected, OV is latched at 0. When overflow (positive or negative) occurs, OV is set to 1 and latched.

- *Test/control flag bit (TC).* TC (bit 11 of status register ST1) is set to 0 or 1 depending on the value of a tested bit. In the case of the NORM instruction, if the exclusive-OR of the two MSBs of the accumulator is true, TC is set to 1.

A number of branch instructions are implemented based on the status of bits C, OV, and TC, and on the value in the accumulator (as compared to zero). For more information about these instructions, see section 5.4, *Conditional Branches, Calls, and Returns*, on page 5-10.

3.3.3 Output Data-Scaling Shifter

The output data-scaling shifter (output shifter) has a 32-bit input connected to the 32-bit output of the accumulator and a 16-bit output connected to the data bus. The shifter copies all 32-bits of the accumulator and then performs a left shift on its content; it can be shifted from zero to seven bits, as specified in the corresponding store instruction. The upper word (SACH instruction) or lower word (SACL instruction) of the shifter is then stored to data memory. The content of the accumulator remains unchanged.

When the output shifter performs the shift, the MSBs are lost and the LSBs are zero filled. Figure 3–7 shows an example in which the accumulator value is shifted left by four bits and the shifted high word is stored to data memory. Figure 3–8 shows the same accumulator value shifted left by 6 bits and then the shifted low word stored.

Figure 3–7. Shifting and Storing the High Word of the Accumulator

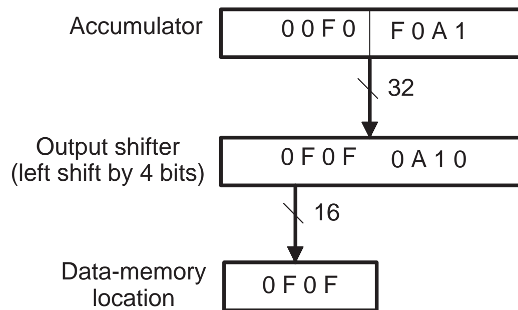
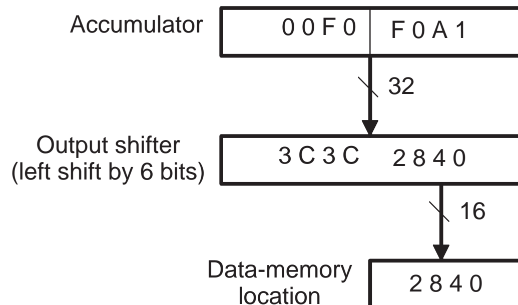


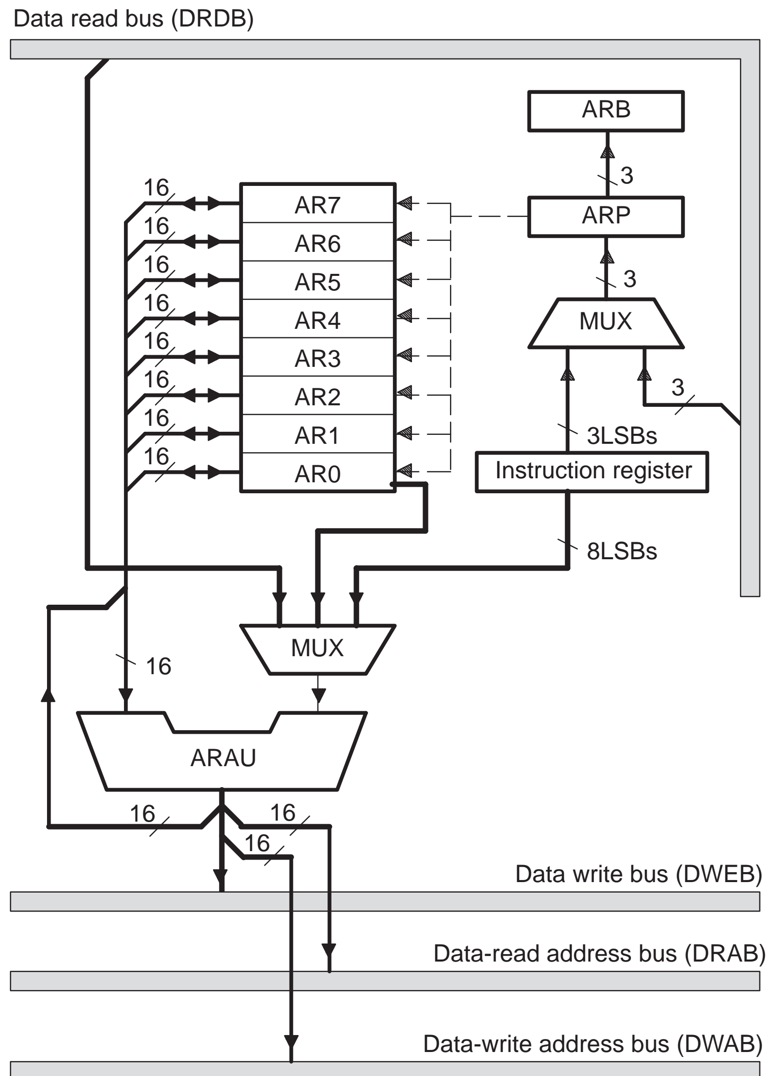
Figure 3–8. Shifting and Storing the Low Word of the Accumulator



3.4 Auxiliary Register Arithmetic Unit (ARAU)

The CPU also contains the auxiliary register arithmetic unit (ARAU), an arithmetic unit independent of the central arithmetic logic unit (CALU). The main function of the ARAU is to perform arithmetic operations on eight auxiliary registers (AR7 through AR0) in parallel with operations occurring in the CALU. Figure 3–9 shows the ARAU and related logic.

Figure 3–9. ARAU and Related Logic



The eight auxiliary registers (AR7–AR0) provide flexible and powerful indirect addressing. Any location in the 64K data memory space can be accessed using a 16-bit address contained in an auxiliary register. For the details of indirect addressing, see section 6.3 on page 6-9.

To select a specific auxiliary register, load the 3-bit auxiliary register pointer (ARP) of status register ST0 with a value from 0 through 7. The ARP can be loaded as a primary operation by the MAR instruction (which only performs modifications to the auxiliary registers and the ARP) or by the LST instruction (which can load a data-memory value to ST0 by way of the data read bus, DRDB). The ARP can be loaded as a secondary operation by any instruction that supports indirect addressing.

The register pointed to by the ARP is referred to as the *current auxiliary register* or *current AR*. During the processing of an instruction, the content of the current auxiliary register is used as the address at which the data-memory access will take place. The ARAU passes this address to the data-read address bus (DRAB) if the instruction requires a read from data memory, or it passes the address to the data-write address bus (DWAB) if the instruction requires a write to data memory. After the instruction uses the data value, the contents of the current auxiliary register can be incremented or decremented by the ARAU, which implements unsigned 16-bit arithmetic.

3.4.1 ARAU and Auxiliary Register Functions

The ARAU performs the following operations:

- ❑ Increments or decrements an auxiliary register value by 1 or by an index amount (by way of any instruction that supports indirect addressing)
- ❑ Adds a constant value to an auxiliary register value (ADRK instruction) or subtracts a constant value from an auxiliary register value (SBRK instruction). The constant is an 8-bit value taken from the eight LSBs of the instruction word.
- ❑ Compares the content of AR0 with the content of the current AR and puts the result in the test/control flag bit (TC) of status register ST1 (CMPR instruction). The result is passed to TC by way of the data write bus (DWEB).

Normally, the ARAU performs its arithmetic operations in the decode phase of the pipeline (when the instruction specifying the operations is being decoded). This allows the address to be generated before the decode phase of the next instruction. There is an exception to this rule: During processing of the NORM instruction, the auxiliary register and/or ARP modification is done during the

execute phase of the pipeline. For information on the operation of the pipeline, see section 5.2 on page 5-7.

In addition to using the auxiliary registers to reference data-memory addresses, you can use them for other purposes. For example, you can:

- Use the auxiliary registers to support conditional branches, calls, and returns by using the CMPR instruction. This instruction compares the content of AR0 with the content of the current AR and puts the result in the test/control flag bit (TC) of status register ST1.
- Use the auxiliary registers for temporary storage by using the LAR instruction to load values into the registers and the SAR instruction to store AR values to data memory.
- Use the auxiliary registers as software counters, incrementing or decrementing them as necessary.

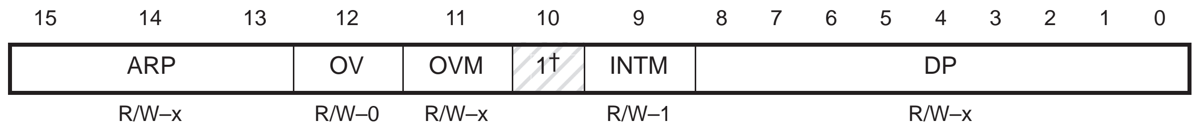
3.5 Status Registers ST0 and ST1

The ’C20x has two status registers, ST0 and ST1, which contain status and control bits. These registers can be stored to and loaded from data memory, thus allowing the status of the machine to be saved and restored for subroutines.

The LST (load status register) instruction writes to ST0 and ST1, and the SST (store status register) instruction reads from ST0 and ST1 (with the exception of the INTM bit, which is not affected by the LST instruction). Many of the individual bits of these registers can be set and cleared using the SETC and CLRC instructions. For example, the sign-extension mode is set with SETC SXM and cleared with CLRC SXM.

Figure 3–10 and Figure 3–11 show the organization of status registers ST0 and ST1, respectively. Several bits in the status registers are reserved; they are always read as logic 1s. The other bits are described in alphabetical order in Table 3–2.

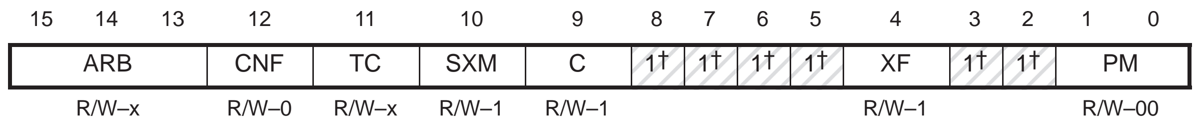
Figure 3–10. Status Register ST0



Note: R = Read access; W = Write access; value following dash (–) is value after reset (x means value not affected by reset).

† This reserved bit is always read as 1. Writes have no effect on it.

Figure 3–11. Status Register ST1



Note: R = Read access; W = Write access; value following dash (–) is value after reset (x means value not affected by reset).

† These reserved bits are always read as 1s. Writes have no effect on them.

Table 3–2. Bit Fields of Status Registers ST0 and ST1

Name	Description
ARB	**Auxiliary register pointer buffer.** Whenever the auxiliary register pointer (ARP) is loaded, the previous ARP value is copied to the ARB, except during an LST (load status register) instruction. When the ARB is loaded by an LST instruction, the same value is also copied to the ARP.
ARP	**Auxiliary register pointer.** This 3-bit field selects which auxiliary register (AR) to use in indirect addressing. When the ARP is loaded, the previous ARP value is copied to the ARB register, except during an LST (load status register) instruction. The ARP may be modified by memory-reference instructions using indirect addressing, and by the MAR (modify auxiliary register) and LST instructions. When the ARB is loaded by an LST instruction, the same value is also copied to the ARP. For more details on the use of ARP in indirect addressing, see section 6.3, *Indirect Addressing Mode*, on page 6-9.
C	**Carry bit.** This bit is set to 1 if the result of an addition generates a carry, or cleared to 0 if the result of a subtraction generates a borrow. Otherwise, it is cleared after an addition or set after a subtraction, except if the instruction is ADD or SUB with a 16-bit shift. In these cases, ADD can only set and SUB only clear the carry bit, but cannot affect it otherwise. The single-bit shift and rotate instructions also affect this bit, as well as the SETC, CLRC, and LST instructions. The conditional branch, call, and return instructions can execute based on the status of C. C is set to 1 on reset.
CNF	**On-chip DARAM configuration bit.** This bit determines whether reconfigurable dual-access RAM blocks are mapped to data space or to program space. The CNF bit may be modified by the SETC CNF, CLRC CNF, and LST instructions. Reset clears the CNF bit to 0. For more information about CNF and the dual-access RAM blocks, see Chapter 4, *Memory and I/O Spaces*. CNF = 0 Reconfigurable dual-access RAM blocks are mapped to data space. CNF = 1 Reconfigurable dual-access RAM blocks are mapped to program space.
DP	**Data page pointer.** When an instruction uses direct addressing, the 9-bit DP field is concatenated with the 7 LSBs of the instruction word to form a full 16-bit data-memory address. For more details, see section 6.2, *Direct Addressing Mode*, on page 6-4. The LST and LDP (load DP) instructions can modify the DP field.
INTM	**Interrupt mode bit.** This bit enables or disables all maskable interrupts. INTM is set and cleared by the SETC INTM and CLRC INTM instructions, respectively. INTM has no effect on the nonmaskable interrupts $\overline{RS}$ and $\overline{NMI}$ or on interrupts initiated by software. INTM is unaffected by the LST (load status register) instruction. INTM is set to 1 when an interrupt trap is taken (except in the case of the TRAP instruction) and at reset. INTM = 0 All unmasked interrupts are enabled. INTM = 1 All maskable interrupts are disabled.
OV	**Overflow flag bit.** This bit holds a latched value that indicates whether overflow has occurred in the CALU. OV is set to 1 when an overflow occurs in the CALU. Once an overflow occurs, the OV bit remains set until it is cleared by a reset, a conditional branch on overflow (OV) or no overflow (NOV), or an LST instruction .

Table 3–2. Bit Fields of Status Registers ST0 and ST1 (Continued)

Name	Description
OVM	**Overflow mode bit.** OVM determines how overflows in the CALU are handled. The SETC and CLRC instructions set and clear this bit, respectively. An LST instruction can also be used to modify OVM. OVM = 0 Results overflow normally in the accumulator. OVM = 1 The accumulator is set to either its most positive or negative value upon encountering an overflow. (See section 3.3.2, *Accumulator.*)
PM	**Product shift mode.** PM determines the amount that the PREG value is shifted on its way to the CALU or to data memory. Note that the content of the PREG remains unchanged; the value is copied to the product shifter and shifted there. PM is loaded by the SPM and LST instructions. The PM bits are cleared by reset. PM = 00 The multiplier's 32-bit product is passed to the CALU or to data memory with no shift. PM = 01 The output of the PREG is left shifted one place (with the LSBs zero filled) before being passed to the CALU or to data memory. PM = 10 The output of the PREG is left shifted four bits (with the LSBs zero filled) before being passed to the CALU or to data memory. PM = 11 This mode produces a right shift of six bits, sign extended.
SXM	**Sign-extension mode bit.** SXM does not affect the basic operation of certain instructions. For example, the ADDS instruction suppresses sign extension regardless of SXM. This bit is set by the SETC SXM instruction and cleared by the CLRC SXM instruction, and may be loaded by the LST instruction. SXM is set to 1 by reset. SXM = 0 This mode suppresses sign extension. SXM = 1 In this mode, data values that are shifted in the input shifter are sign extended before they are passed to the CALU.
TC	**Test/control flag bit.** The TC bit is set to 1 if a bit tested by BIT or BITT is a 1, if a compare condition tested by CMPR exists between the current auxiliary register and AR0, or if the exclusive-OR function of the two MSBs of the accumulator is true when tested by a NORM instruction. The conditional branch, call, and return instructions can execute based on the condition of the TC bit. The TC bit is affected by the BIT, BITT, CMPR, LST, and NORM instructions.
XF	**XF pin status bit.** This bit determines the state of the XF pin, which is a general-purpose output pin. XF is set by the SETC XF instruction and cleared by the CLRC XF instruction. XF can also be modified with an LST instruction. XF is set to 1 by reset.

Chapter 4

Memory and I/O Spaces

This chapter describes the 'C20x memory configuration options and the address maps of the individual 'C20x devices. It also illustrates typical ways of interfacing the 'C20x with external memory and external input/output (I/O) devices.

Each 'C20x device has a 16-bit address line that accesses four individually selectable spaces (224K words total):

- A 64K-word program space
- A 64K-word local data space
- A 32K-word global data space
- A 64K-word I/O space

Also available on select 'C20x devices are an on-chip bootloader and a HOLD operation. The on-chip bootloader allows a 'C20x to boot software from an external source to a 16-bit external RAM at reset. The HOLD operation allows a 'C20x to give external devices direct memory access to external program, data, and I/O spaces.

Topic **Page**

4.1 Overview of the Memory and I/O Spaces 4-2
4.2 Program Memory 4-5
4.3 Local Data Memory 4-7
4.4 Global Data Memory 4-11
4.5 I/O Space 4-14
4.6 Direct Memory Access Using the HOLD Operation 4-18
4.7 Device-Specific Information 4-22
4.8 'C203 Bootloader 4-30
4.9 'C206/LC206 Bootloader 4-39

4.1 Overview of the Memory and I/O Spaces

The ’C20x address map is organized into four individually selectable spaces:

- Program memory (64K words) contains the instructions to be executed, as well as immediate data used during program execution.
- Local data memory (64K words) holds data used by the instructions.
- Global data memory (32K words) shares data with other processors or serves as additional data space. Addresses in the upper 32K words (8000h–FFFFh) of local data memory can be used for global data memory.
- Input/output (I/O) space (64K words) interfaces to external peripherals and contains registers for the on-chip peripherals.

These spaces provide a total address range of 224K words. The ’C20x includes a considerable amount of on-chip memory to aid in system performance and integration and a considerable amount of addresses that can be used for external memory and I/O devices.

The advantages of operating from on-chip memory are:

- Higher performance than external memory (because the wait states required for slower external memories are avoided)
- Lower cost than external memory
- Lower power consumption than external memory

The advantage of operating from external memory is the ability to access a larger address space.

The ’C20x design is based on an enhanced Harvard architecture. The ’C20x memory spaces are accessible on three parallel buses—the program address bus (PAB), the data-read address bus (DRAB), and the data-write address bus (DWAB). Because the operations of the three buses are independent, it is possible to access both the program and data spaces simultaneously. Within a given machine cycle, the central arithmetic logic unit (CALU) can execute as many as three concurrent memory operations.

4.1.1 Pins for Interfacing to External Memory and I/O Spaces

Four pin types are used for interfacing to external memory and I/O space. Table 4–1 describes the main types as:

- External buses. Sixteen signals (A15–A0) are available for passing an address from the 'C20x to another device. Sixteen signals (D15–D0) are available for transferring a data value between the 'C20x and another device.
- Select signals. These signals can be used by external devices to determine when the 'C20x is requesting access to off-chip locations, and whether that request is for data, program, global, or I/O space.
- Read/write signals. These signals indicate to external devices the direction of a data transfer (to the 'C20x or from the 'C20x).
- Request/control signals. The input request signals (BOOT, MP/MC, RAMEN, READY, and HOLD) effect a change in the operation of the 'C20x. The output HOLDA is the response to HOLD.

Table 4–1. Pins for Interfacing With External Memory and I/O Spaces

	Pin(s)	Description
External buses	A15–A0	The 16 lines of the external address bus. This bus can address up to 64K words of external memory or I/O space.
	D15–D0	The 16 bidirectional lines of the external data bus. This bus carries data to and from external memory or I/O space.
Select signals	DS	Data memory select pin. The 'C20x asserts DS to indicate an access to external data memory (local or global).
	BR	Bus request pin. The 'C20x asserts both BR and DS to indicate an access to global data memory.
	PS	Program memory select pin. The 'C20x asserts PS to indicate an access to external program memory.
	IS	I/O space select pin. The 'C20x asserts IS to indicate an access to external I/O space.
	STRB	External access active strobe. The 'C20x asserts STRB during accesses to external program, data, or I/O space.

Table 4–1. Pins for Interfacing With External Memory and I/O Spaces (Continued)

	Pin(s)	Description
Read/write signals	R/$\overline{\text{W}}$	Read/write pin. This pin indicates the direction of transfer between the ’C20x and external program, data, or I/O space.
	$\overline{\text{RD}}$	Read select pin. The ’C20x asserts $\overline{\text{RD}}$ to request a read from external program, data, or I/O space.
	$\overline{\text{WE}}$	Write enable pin. The ’C20x asserts $\overline{\text{WE}}$ to request a write to external program, data, or I/O space.
Request/control signals	$\overline{\text{BOOT}}$	Boot-load pin. This pin is only on devices that have the on-chip bootloader. If $\overline{\text{BOOT}}$ is low during a hardware reset, the ’C20x transfers code from EPROM in global data memory to RAM in external program memory.
	MP/$\overline{\text{MC}}$	Microprocessor/microcomputer pin. This pin is only on devices with on-chip non-volatile program memory. The level on this pin is tested at reset. If MP/$\overline{\text{MC}}$ is high, the device is in microprocessor mode (the reset vector is fetched from external memory). If MP/$\overline{\text{MC}}$ is low, the device is in microcomputer mode (the reset vector is fetched from on-chip memory).
	RAMEN	Single-access RAM enable pin. On ’C20x devices with on-chip single-access RAM, when this pin is high, the RAM is enabled; when this pin is low, the RAM is disabled.
	READY	External device ready pin (for generating wait states externally). When this pin is driven low, the ’C20x waits one CPU cycle and then tests READY again. After READY is driven low, the ’C20x does not continue processing until READY is driven high. If READY is not used, it should be kept high. For a ’C20x device with a bootloader, READY must be high at boot time.
	$\overline{\text{HOLD}}$	HOLD operation request pin. An external device can request control of the external buses by asserting $\overline{\text{HOLD}}$. After the ’C20x (along with proper software logic) asserts $\overline{\text{HOLDA}}$, the external device controls the buses until it deasserts $\overline{\text{HOLD}}$.
	$\overline{\text{HOLDA}}$	$\overline{\text{HOLD}}$ acknowledge pin. The ’C20x (with assistance from proper program code) asserts $\overline{\text{HOLDA}}$ to acknowledge that $\overline{\text{HOLD}}$ has been asserted and places its external buses in high impedance.

4.2 Program Memory

Program-memory space holds the code for applications; it can also hold table information and constant operands. The program-memory space addresses up to 64K 16-bit words. Every ’C20x device contains a DARAM block B0 that can be configured as program memory or data memory. Other on-chip program memory may be SARAM and ROM or flash memory. For information on configuring on-chip program-memory blocks, see section 4.7.

4.2.1 Interfacing With External Program Memory

The ’C20x can address up to 64K words of external program memory. While the ’C20x is accessing the on-chip program-memory blocks, the external memory signals $\overline{PS}$ and $\overline{STRB}$ are in the high state. The external buses are active only when the ’C20x is accessing locations within the address ranges mapped to external memory. An active $\overline{PS}$ signal indicates that the external buses are being used for program memory. Whenever the external buses are active (when external memory or I/O space is being accessed), the ’C20x drives the $\overline{STRB}$ signal low.

For fast memory interfacing, it is important to select external memory with fast access time. If fast memory is not available, or if speed is not a serious consideration, you can use the the READY signal and/or the on-chip wait-state generator to create wait states.

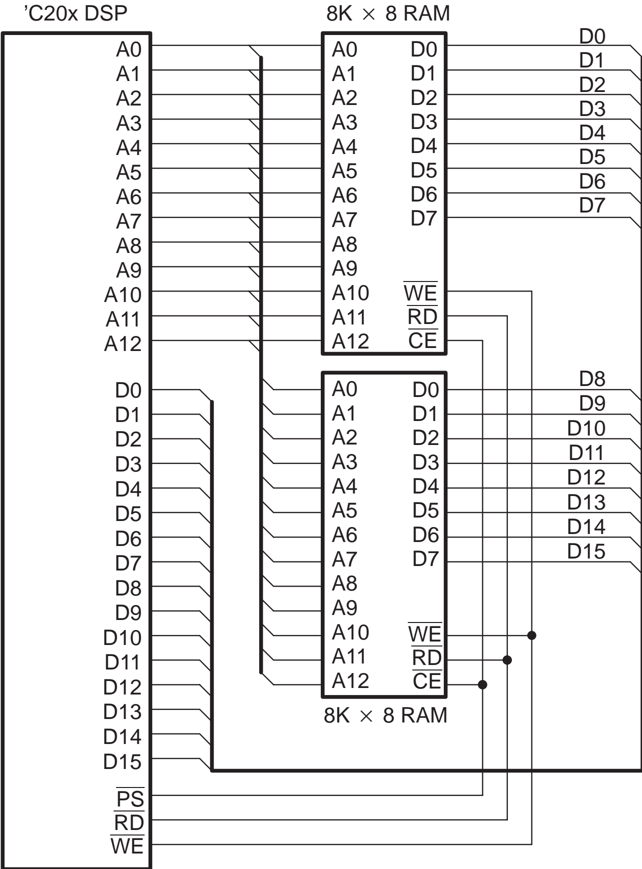
Figure 4–1 shows an example of interfacing to external program memory. In the figure, 8K × 16-bit static memory is interfaced to the ’C20x using two 8K × 8-bit RAMs.

CAUTION

Obtain the Proper Timing Information

When interfacing memory with high-speed ’C20x devices, refer to the data sheet for that ’C20x device for the required access, delay, and hold times.

Figure 4–1. Interface With External Program Memory



4.3 Local Data Memory

The local data-memory space addresses up to 64K 16-bit words. Every ’C20x device has three on-chip DARAM blocks: B0, B1, and B2. Block B0 has 256 words that are configurable as either data locations or program locations. Blocks B1 (256 words) and B2 (32 words) have a total of 288 words that are available for data memory only. Some ’C20x devices, in addition to the three DARAM blocks, have an on-chip SARAM block that can be used for program and/or data memory. Section 4.7 tells how to configure these memory blocks.

Data memory can be addressed with either of two addressing modes: direct-addressing mode or indirect-addressing mode. Addressing modes are described in detail in Chapter 6.

When direct addressing is used, data memory is addressed in blocks of 128 words called data pages. Figure 4–2 shows how these blocks are addressed. The entire 64K of data memory consists of 512 data pages labeled 0 through 511. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register ST0. Each of the 128 words on the current page is referenced by a 7-bit offset, which is taken from the instruction that is using direct addressing. Therefore, when an instruction uses direct addressing, you must specify both the data page (with a preceding instruction) and the offset (in the instruction that accesses data memory).

Figure 4–2. Pages of Data Memory

DP value	Offset	’C20x Data Memory
0000 0000 0 ⋮ 0000 0000 0	000 0000 ⋮ 111 1111	Page 0: 0000h–007Fh
0000 0000 1 ⋮ 0000 0000 1	000 0000 ⋮ 111 1111	Page 1: 0080h–00FFh
0000 0001 0 ⋮ 0000 0001 0	000 0000 ⋮ 111 1111	Page 2: 0100h–017Fh
. . .	. . .	. . .
1111 1111 1 ⋮ 1111 1111 1	000 0000 ⋮ 111 1111	Page 511: FF80h–FFFFh

4.3.1 Data Page 0 Address Map

Table 4–2 shows the address map of data page 0 (addresses 0000h–007Fh). Note the following:

- Three memory-mapped registers can be accessed with zero wait states:
  - Interrupt mask register (IMR)
  - Global memory allocation register (GREG)
  - Interrupt flag register (IFR)
- The test/emulation reserved area is used by the test and emulation systems for special information transfers.

> **CAUTION**
>
> **Do Not Write to Test/Emulation Addresses**
>
> **Writing to the test/emulation addresses can cause the device to change its operational mode and, therefore, affect the operation of an application.**

- The scratch-pad RAM block (B2) includes 32 words of DARAM that provide for variable storage without fragmenting the larger RAM blocks, whether internal or external. This RAM block supports dual-access operations and can be addressed with any data-memory addressing mode.

Table 4–2. Data Page 0 Address Map

Address	Name	Description
0000h–0003h	–	Reserved
0004h	IMR	Interrupt mask register
0005h	GREG	Global memory allocation register
0006h	IFR	Interrupt flag register
0023h–0027h	–	Reserved
002Bh–002Fh	–	Reserved for test/emulation
0060h–007Fh	B2	Scratch-pad RAM (DARAM B2)

4.3.2 Interfacing With External Local Data Memory

While the ’C20x is accessing the on-chip local data-memory blocks and memory-mapped registers, the external memory signals $\overline{\text{DS}}$ and $\overline{\text{STRB}}$ are in the high state. The external buses are active only when the ’C20x is accessing locations within the address ranges mapped to external memory. An active $\overline{\text{DS}}$ signal indicates that the external buses are being used for data memory. Whenever the external buses are active (when external memory or I/O space is being accessed) the ’C20x drives the $\overline{\text{STRB}}$ signal low.

For fast memory interfacing, it is important to select external memory with fast access time. If fast memory is not available, or if speed is not a serious consideration, you can use the the READY signal and/or the on-chip wait-state generator to create wait states.

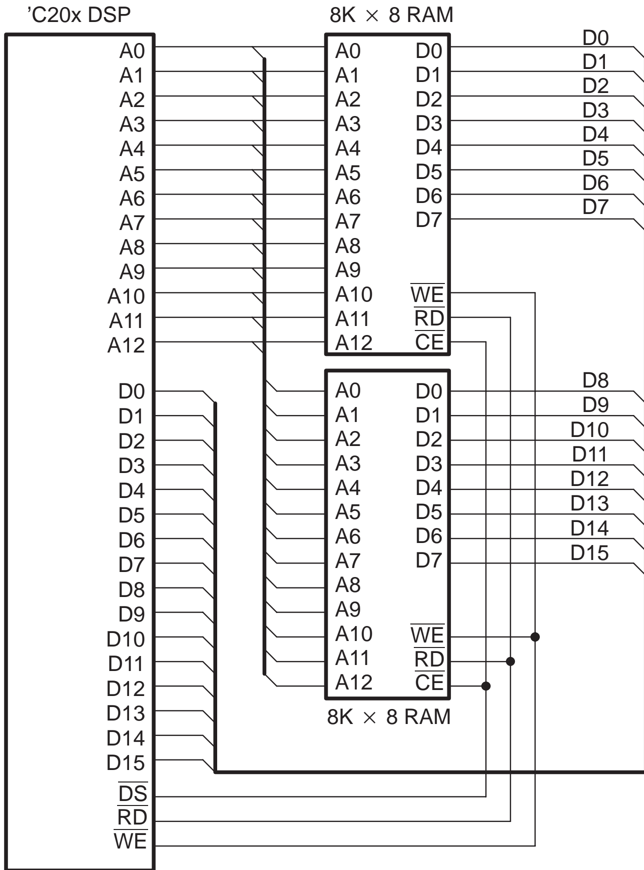
Figure 4–3 shows an example of interfacing to external data memory. In the figure 8K × 16-bit static memory is interfaced to the ’C20x using two 8K × 8-bit RAMs. The RAM devices must have fast access times if the internal instruction speed is to be maintained.

CAUTION

Obtain the Proper Timing Information

When interfacing memory with high-speed ’C20x devices, refer to the data sheet for that ’C20x device for the required access, delay, and hold times.

Figure 4–3. Interface With External Local Data Memory



4.4 Global Data Memory

Addresses in the upper 32K words (8000h–FFFFh) of local data memory can be used for global data memory. The global memory allocation register (GREG) determines the size of the global data-memory space, which is between 0 and 32K words. The GREG is connected to the eight LSBs of the internal data bus and is memory-mapped to data-memory location 0005h. Table 4–3 shows the allowable GREG values and shows the corresponding address range set aside for global data memory. Any remaining addresses within 8000h–FFFFh are available for local data memory.

Note:

Choose only the GREG values listed in Table 4–3. Other values lead to fragmented memory maps.

Table 4–3. Global Data Memory Configurations

GREG Value		Local Memory		Global Memory	
High Byte	**Low Byte**	**Range**	**Words**	**Range**	**Words**
XXXX XXXX	0000 0000	0000h–FFFFh	65 536	–	0
XXXX XXXX	1000 0000	0000h–7FFFh	32 768	8000h–FFFFh	32 768
XXXX XXXX	1100 0000	0000h–BFFFh	49 152	C000h–FFFFh	16 384
XXXX XXXX	1110 0000	0000h–DFFFh	57 344	E000h–FFFFh	8 192
XXXX XXXX	1111 0000	0000h–EFFFh	61 440	F000h–FFFFh	4 096
XXXX XXXX	1111 1000	0000h–F7FFh	63 488	F800h–FFFFh	2 048
XXXX XXXX	1111 1100	0000h–FBFFh	64 512	FC00h–FFFFh	1 024
XXXX XXXX	1111 1110	0000h–FDFFh	65 024	FE00h–FFFFh	512
XXXX XXXX	1111 1111	0000h–FEFFh	65 280	FF00h–FFFFh	256

Note: X = Don’t care

As an example of configuring global memory, suppose you want to designate 8K addresses as global addresses. You would write the 8-bit value 11100000_2 to the eight LSBs of the GREG (see Figure 4–4). This would designate addresses E000h–FFFFh of data memory as global data addresses (see Figure 4–5).

Figure 4–4. GREG Register Set to Configure 8K for Global Data Memory

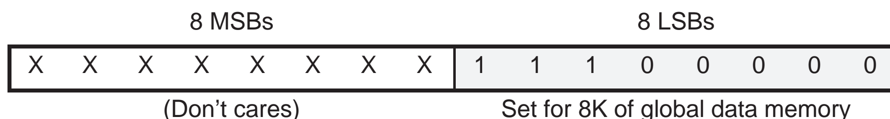
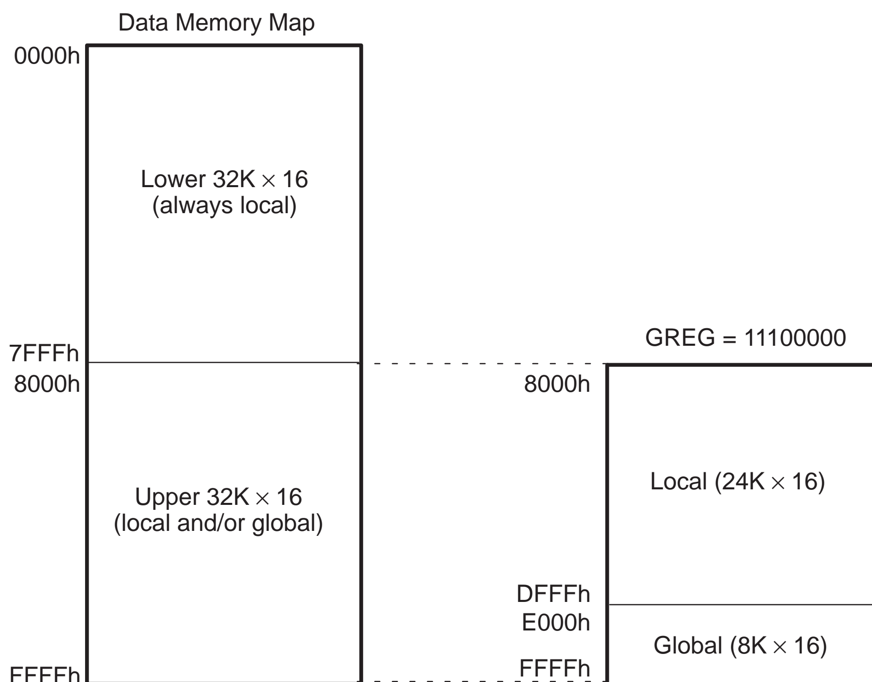


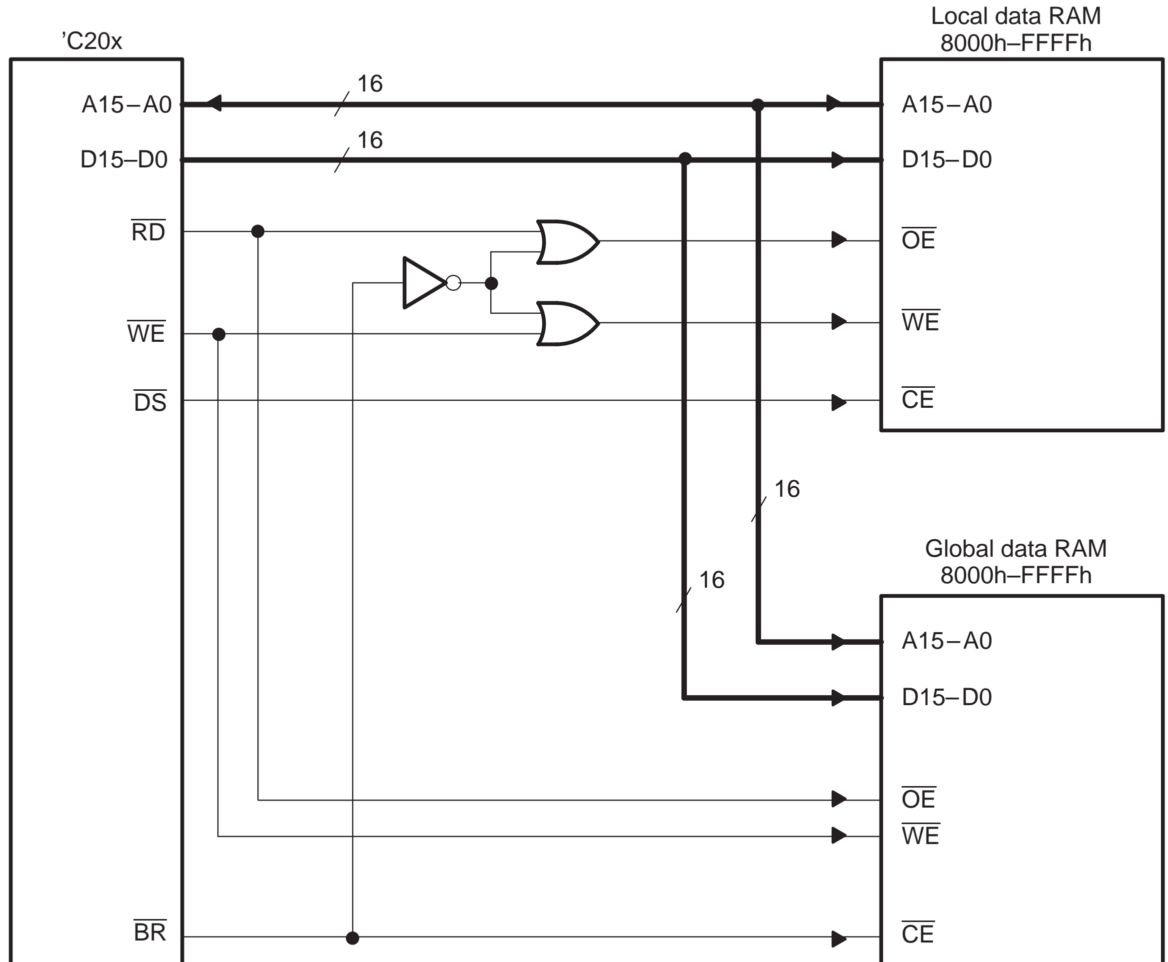
Figure 4–5. Global and Local Data Memory for GREG = 11100000



4.4.1 Interfacing With External Global Data Memory

When a program accesses any data-memory address, the ’C20x drives the $\overline{DS}$ signal low. If that address is within a range defined by the GREG as global, $\overline{BR}$ and $\overline{DS}$ are asserted. Because $\overline{BR}$ differentiates local and global accesses, you can use the GREG to extend data memory by up to 32K. Figure 4–6 shows two external RAMs that are sharing data-memory addresses 8000h–FFFFh. Overlapping addresses must be reconfigured with the GREG in order to be toggled between local memory and global memory. For example, in the system of Figure 4–6, when GREG = $XXXXXXXX00000000_2$ (no global memory), the local data RAM is fully accessible; when GREG = $XXXXXXXX10000000_2$ (all global memory), the local data RAM is not accessible.

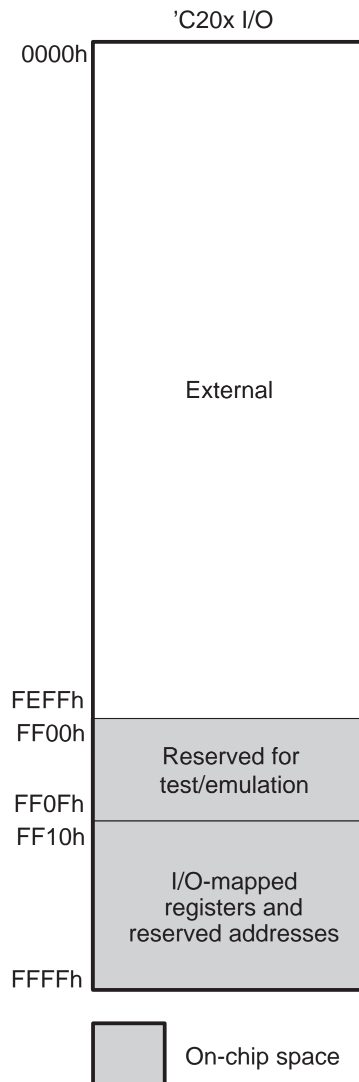
Figure 4–6. Using 8000h–FFFFh for Local and Global External Memory



4.5 I/O Space

The ’C20x supports an I/O address range of 64K 16-bit words. Figure 4–7 shows the ’C20x I/O address map.

Figure 4–7. I/O Address Map for the ’C20x



The map has three main sections of addresses:

- Addresses 0000h–FEFFh allow access to off-chip peripherals typically used in DSP applications, such as digital-to-analog and analog-to-digital converters.
- Addresses FF00h–FF0Fh are mapped to on-chip I/O space. These addresses are reserved for test purposes and should not be used.
- Addresses FF10h–FFFFh are also mapped to on-chip I/O space. These addresses are used for other reserved space and for the on-chip I/O-mapped registers. For 'C20x devices other than the 'C209, Table 4–4 lists the registers mapped to on-chip I/O space. For the I/O-mapped registers on the 'C209, see section 11.2, on page 11-5.

CAUTION

Do Not Write to Reserved Addresses

To avoid unpredictable operation of the processor, do not write to I/O addresses FF00h–FF0Fh or any reserved I/O address in the range FF10–FFFFh (that is, any address not designated for an on-chip peripheral.)

Table 4–4. On-Chip Registers Mapped to I/O Space

I/O Address	Name	Description
FFE4h	PMST	Program memory status register
FFE8h	CLK	CLK register
FFECh	ICR	Interrupt control register
FFF0h	SDTR	Synchronous serial port transmit and receive register
FFF1h	SSPCR	Synchronous serial port control register
FFF2h	SSPST	Synchronous serial port status register
FFF3h	SSPMC	Synchronous serial port multichannel register
FFF4h	ADTR	Asynchronous serial port transmit and receive register
FFF5h	ASPCR	Asynchronous serial port control register
FFF6h	IOSR	Input/output status register
FFF7h	BRD	Baud rate divisor register
FFF8h	TCR	Timer control register
FFF9h	PRD	Timer period register
FFFAh	TIM	Timer counter register
FFFBh	SSPCT	Synchronous serial port counter register
FFFCh	WSGR	Wait-state generator control register

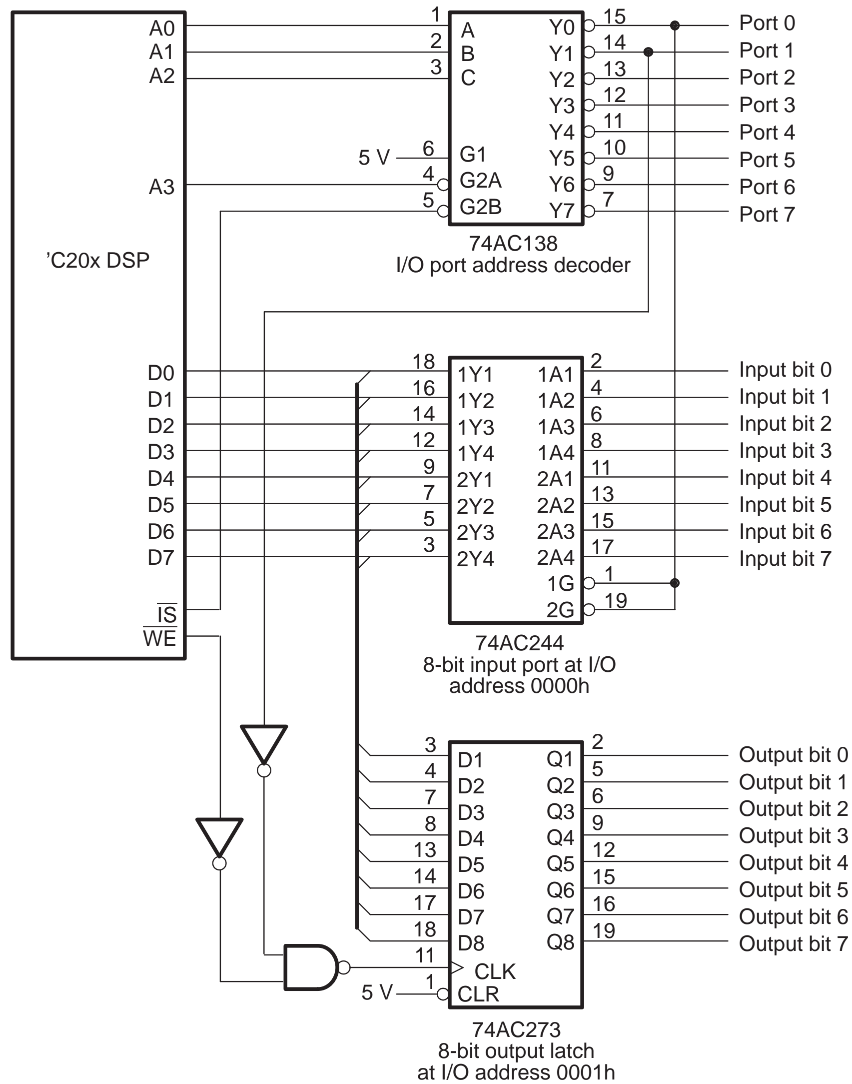
Note: This table does not apply to the ’C209. For the I/O-mapped registers on the ’C209, see section 11.2 on page 11-5.

4.5.1 Accessing I/O Space

All I/O words (external I/O ports and on-chip I/O registers) are accessed with the IN and OUT instructions. Accesses to external parallel I/O ports are multiplexed over the same address and data buses for program and data-memory accesses. These accesses are distinguished from external program and data-memory accesses by $\overline{\text{IS}}$ going low. The data bus is 16 bits wide; however, if you use 8-bit peripherals, you can use either the higher or lower eight lines of the data bus to suit a particular application.

You can use $\overline{\text{RD}}$ with chip-select logic to generate an output-enable signal for an external peripheral. You can also use the $\overline{\text{WE}}$ signal with chip-select logic to generate a write-enable signal for an external peripheral. As an example of interfacing to external I/O space, Figure 4–8 shows interface circuitry for eight input bits and eight output bits. Note that the decode section is simplified if fewer I/O ports are used.

Figure 4–8. I/O Port Interface Circuitry



4.6 Direct Memory Access Using the HOLD Operation

The ’C20x HOLD operation allows direct-memory access to external program, data, and I/O spaces. The process is controlled by two signals:

- **HOLD.** An external device can drive the HOLD/INT1 pin low to request control over the external buses. If the HOLD/INT1 interrupt line is enabled, this triggers an interrupt.
- **HOLDA.** In response to a HOLD interrupt, software logic can cause the processor to issue a HOLD acknowledge (HOLDA pin low), to indicate that it is relinquishing control of its external lines. Upon HOLDA, the external address signals (A15–A0), data signals (D15–D0), and memory-control signals (PS, DS, BR, IS, STRB, R/W, RD, WE) are placed in high impedance.

Following a negative edge on the HOLD/INT1 pin, if interrupt line HOLD/INT1 is enabled, the CPU branches to address 0002h (this branch could also be accomplished with an INTR 1 instruction). Here the CPU fetches the interrupt vector and follows it to the interrupt service routine. If you wish to use this routine for HOLD operations and also for the interrupt INT1, the tasks carried out by this routine will depend on the value of the MODE bit:

- MODE = 1. When the CPU detects a negative edge on HOLD/INT1, it finishes executing the current instruction (or repeat operation) and then forces program control to the interrupt service routine. The interrupt service routine, after successfully testing for MODE = 1, performs the tasks for INT1.
- MODE = 0**.** Interrupt line INT1 is both negative- and positive-edge sensitive. When the CPU detects the negative edge, it finishes executing the current instruction (or repeat operation) and then forces program control to the interrupt service routine. This routine, after successfully testing for MODE = 0, executes an IDLE instruction. Upon IDLE, HOLDA is asserted and the external lines are placed in high impedance. Only after detecting a rising edge on the HOLD/INT1 pin, the CPU exits the IDLE state, deasserts HOLDA, and returns the external lines to their normal states.

Example 4–1 shows an interrupt service routine that tests the MODE bit and acts accordingly. Note that the IDLE instruction should be placed inside the interrupt service routine to issue HOLDA. Also note that the interrupt program code disables all maskable interrupts except HOLD/INT1 to allow safe recovery of HOLDA and the buses. Any other sequence of CPU code will cause undesirable bus control and is not recommended. (Interrupt operation is explained in detail in section 5.6 on page 5-15.)

Example 4–1. An Interrupt Service Routine Supporting $\overline{INT1}$ and $\overline{HOLD}$

```
          .mmregs               ;Include c2xx memory-mapped registers.
ICR       .set   0FFECh         ;Define interrupt control register in I/O space.
ICRSHDW   .set   060h           ;Define ICRSHDW in scratch pad location.

*  Interrupt vectors  *

reset     B      main           ;0 – reset , Branch to main program on reset.
Int1h     B      int1_hold      ;1 – external interrupt 1 or HOLD.
          .space 40*16          ;Fill 0000 between vectors and main program.
main:     SPLK   #0001h,imr     ;Enable HOLD/INT1 interrupt line.
          CLRC   INTM
wait:     B      wait

*********Interrupt service routine for HOLD logic*****************************

int1_hold:
          ; Perform any desired context save.

          LDP    #0             ;Set data-memory page to 0.
          IN     ICRSHDW, ICR   ;Save the contents of ICR register.
          LACL   #010h          ;Load accumulator (ACC) with mask for MODE bit.
          AND    ICRSHDW        ;Filter out all bits except MODE bit.
          BCND   int1, neq      ;Branch if MODE bit is 1, else in HOLD mode.
          LACC   imr, 0         ;Load ACC with interrupt mask register.
          SPLK   #1, imr        ;Mask all interrupts except interrupt1/HOLD.
          IDLE                  ;Enter HOLD mode. Issues HOLDA, and puts
                                ;buses in high impedance. Wait until
                                ;rising edge is seen on HOLD/INT1 pin.
          SPLK   #1, ifr        ;Clear HOLD/INT1 flag in interrupt flag register
                                ;to prevent re-entering HOLD mode.
          SACL   imr            ;Restore interrupt mask register.

          ; Perform necessary context restore.

          CLRC   INTM           ;Enable all interrupts.
          RET                   ;Return from HOLD interrupt.

int1:     NOP                   ;Replace these NOPs with desired int1 interrupt
          NOP                   ;service routine.
                                ; Perform necessary context restore.
          CLRC   INTM           ;Enable all interrupts.
          RET                   ;Return from interrupts.
```

Here are three valid methods for exiting the IDLE state, thus deasserting HOLDA and restoring the buses to normal operation:

- ❑ Cause a rising edge on the HOLD/INT1 pin when MODE = 0.
- ❑ Assert system reset at the reset pin.
- ❑ Assert the nonmaskable interrupt NMI at the NMI pin.

If reset or NMI occurs while HOLDA is asserted, the CPU will deassert HOLDA regardless of the level on the HOLD/INT1 pin. Therefore, to avoid further conflicts in bus control, the system hardware logic should restore HOLD to a high state.

4.6.1 HOLD During Reset

The HOLD logic can be used to put the buses in a high-impedance state at power-on or reset. This feature is useful in extending the DSP memory control to external processors. If HOLD is driven low during reset, normal reset operation occurs internally, but HOLDA will be asserted, placing all buses and control lines in a high-impedance state. Upon release of both HOLD and RS, execution starts from program location 0000h.

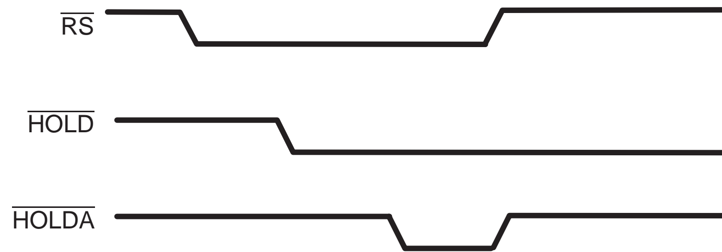
Either of the following conditions will cause the processor to deassert HOLDA and return the buses to a normal state:

- ❑ HOLD is deasserted before reset is deasserted. See Figure 4–9. This is the normal recovery condition after a HOLD operation. After the HOLD signal goes high, the HOLDA signal will be deasserted, and the buses will assume normal states.

Figure 4–9. HOLD Deasserted Before Reset Deasserted

- Reset is deasserted before $\overline{\text{HOLD}}$ is deasserted. See Figure 4–10. The CPU will deassert $\overline{\text{HOLDA}}$ regardless of the $\overline{\text{HOLD}}$ signal after the 16 clock cycles required for normal reset operation. Along with the $\overline{\text{HOLDA}}$ signal, the buses will assume normal states. The external system hardware logic should restore the $\overline{\text{HOLD}}$ signal to a high state to avoid conflicts in HOLD logic.

Figure 4–10. Reset Deasserted Before $\overline{\text{HOLD}}$ Deasserted



4.7 Device-Specific Information

For ’C20x devices other than the ’C209, this section mentions the presence or absence of the bootloader and HOLD features, shows address maps, and explains the contents and configuration of the program-memory and data-memory maps. For details about the memory and I/O spaces of the ’C209, see section 11.2 on page 11-5.

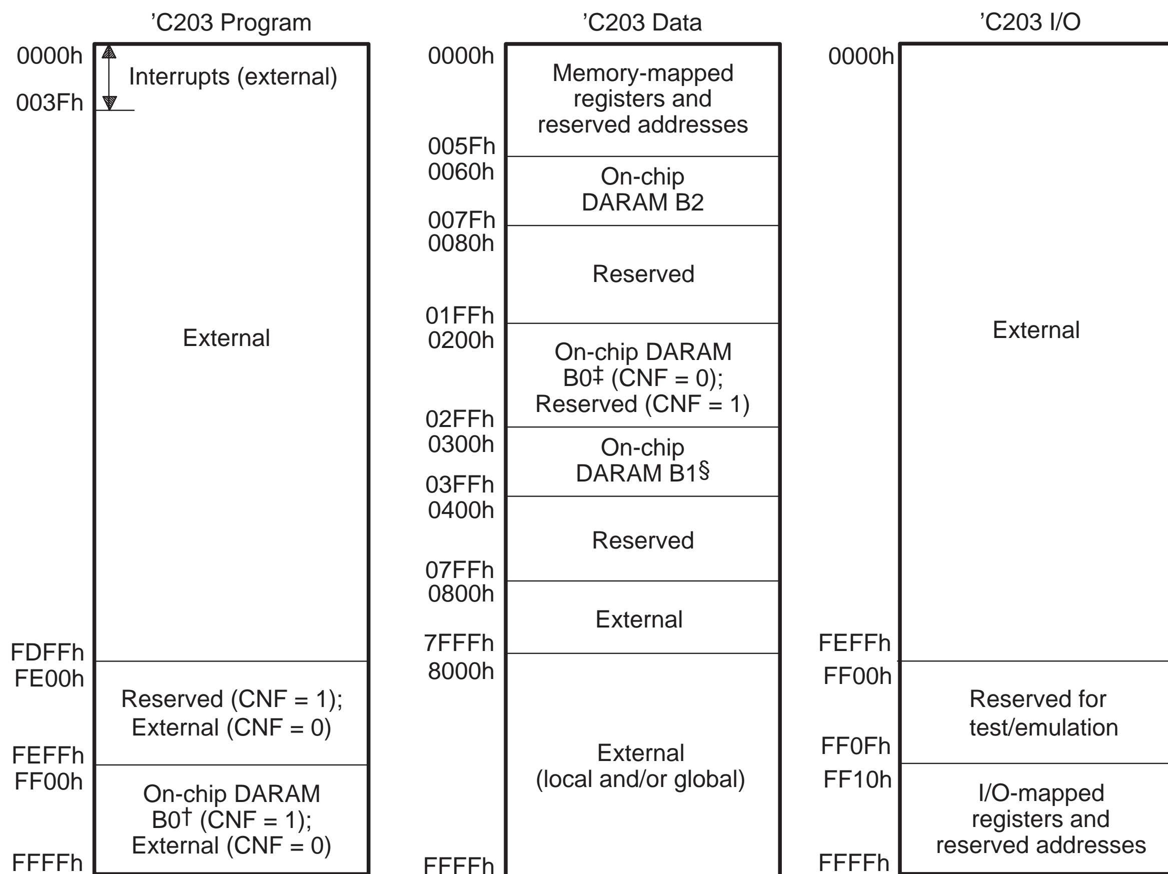
4.7.1 TMS320C203 Address Maps and Memory Configuration

The ’C203 has a ’C20x on-chip bootloader and supports the ’C20x HOLD operation. Figure 4–11 shows the ’C203 address map.

The on-chip program and data memory available on the ’C203 consists of:

- ❑ DARAM B0 (256 words, for program or data memory)
- ❑ DARAM B1 (256 words, for data memory)
- ❑ DARAM B2 (32 words, for data memory)

Figure 4–11.’C203 Address Map



† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to here as reserved when CNF = 1.

‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to here as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to here as reserved.

DARAM blocks B1 and B2 are fixed, but DARAM block B0 may be mapped to program space or data space, depending on the value of the CNF bit (bit 12 of status register ST1):

- CNF = 0**.** B0 is mapped to data space and is accessible at data addresses 0200h–02FFh. Note that the addressable external *program* memory increases by 512 words.
- CNF = 1. B0 is mapped to program space and is accessible at program addresses FF00h–FFFFh.

At reset, CNF = 0.

Table 4–5 shows the program-memory options for the ’C203; Table 4–6 lists the data-memory options. Note these facts:

- Program-memory addresses 0000h–003Fh are used for the interrupt vectors.
- Data-memory addresses 0000h–005Fh contain on-chip memory-mapped registers and reserved memory.
- Two other on-chip data-memory ranges are always reserved: 0080h–01FFh and 0400h–07FFh.

CAUTION

Do Not Write to Reserved Addresses

To avoid unpredictable operation of the processor, do not write to any addresses labeled Reserved. This includes any data-memory address in the range 0000h–005Fh that is not designated for an on-chip register and any I/O address in the range FF00h–FFFFh that is not designated for an on-chip register.

Table 4–5. ’C203 Program-Memory Configuration Options

CNF	DARAM B0	External	Reserved
0	–	0000h–FFFFh	–
1	FF00h–FFFFh	0000h–FDFFh	FE00h–FEFFh

Table 4–6. ’C203 Data-Memory Configuration Options

CNF	DARAM B0 (hex)	DARAM B1 (hex)	DARAM B2 (hex)	External (hex)	Reserved (hex)
0	0200–02FF	0300–03FF	0060–007F	0800–FFFF	0000–005F 0080–01FF 0400–07FF
1	–	0300–03FF	0060–007F	0800–FFFF	0000–005F 0080–02FF 0400–07FF

4.7.2 TMS320C206/LC206 Address Maps and Memory Configuration

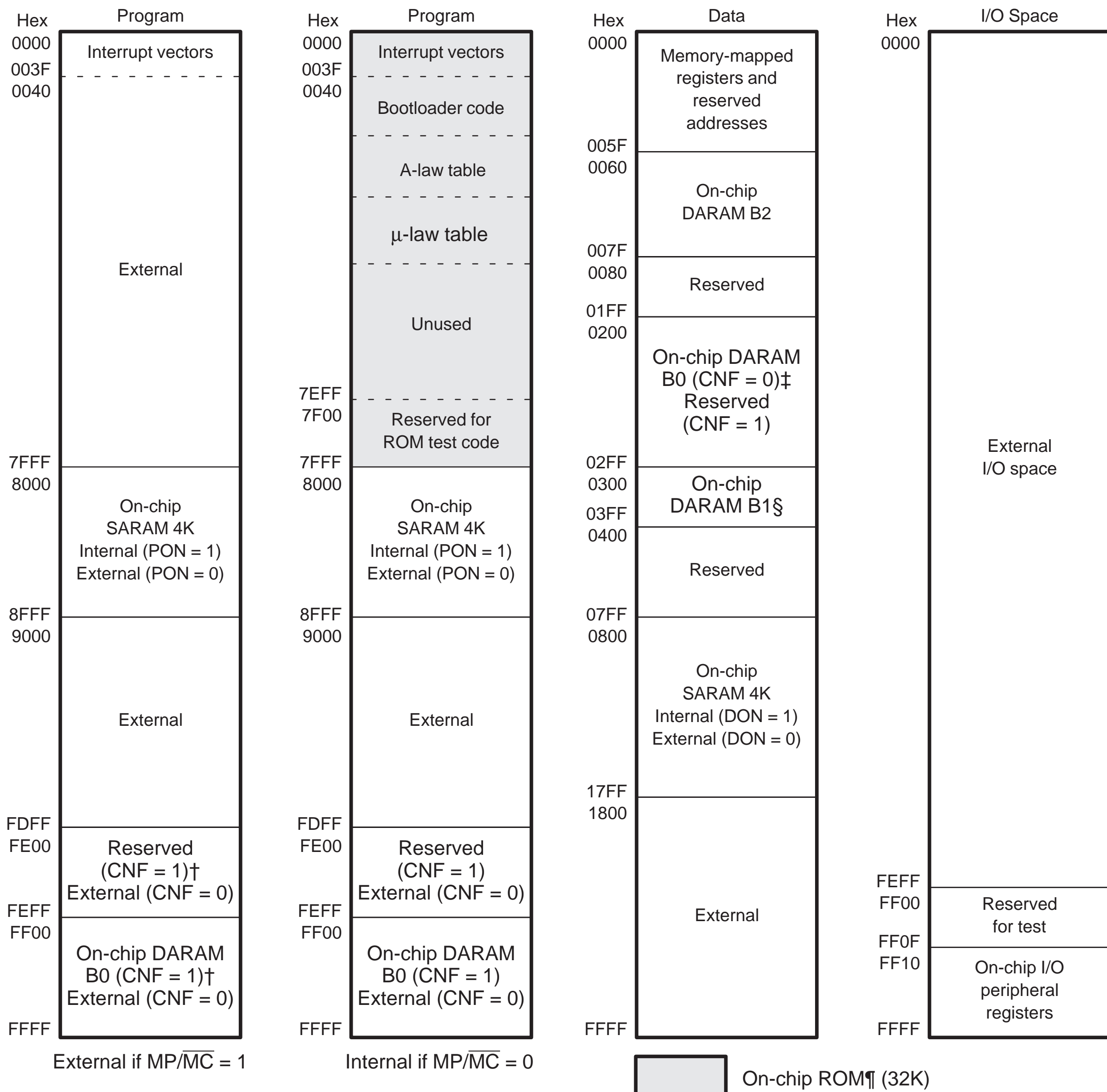
The ’C206/’LC206 have an on-chip bootloader in ROM. Figure 4–12 shows addresses for the ’C206/’LC206 memory map. The on-chip program and data memory available on the ‘C206/’LC206 consists of:

- ROM (32K words, for program memory)
- DARAM B0 (256 words, for program or data memory)
- DARAM B1 (256 words, for data memory)
- DARAM B2 (32 words, for data memory)

The ’C206/’LC206 includes 544 x 16 words of dual-access RAM (DARAM), 4K x 16 single-access RAM (SARAM), and 32K x 16 program ROM memory. The PON and DON bits select the SARAM (4K) mapping in program, data or both. At reset, these bits are 11, mapping the SARAM in both program and data memory.

At reset, if the MP/$\overline{\text{MC}}$ is held high, the device is in microprocessor mode and the program address branches to 0000h (external program space). The MP/$\overline{\text{MC}}$ pin status is latched in the PMST register (bit 0). As long as this bit remains high, the device is in microprocessor mode. PMST register bits can be read and modified in software. If bit 0 is written 0, the device enters microcomputer mode and transfers control to the on-chip ROM at 0000h.

Figure 4–12. TMS320C206, TMS320LC206 Memory Map Configurations



† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to here as reserved when CNF = 1.

‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to here as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. Addresses 0400h–04FFh are referred to here as reserved.

¶ Standard ROM devices will come with boot code and the A-law, µ-law table.

4.7.3 TMS320F206 Address Maps and Memory Configuration

The ’F206 has an on-chip serial loader in flash EEPROM. Figure 4–13 shows addresses for the ‘F206 memory map. The on-chip program and data memory available on the ‘F206 consists of:

- ❑ Flash EEPROM (32K words, for program memory)
- ❑ DARAM B0 (256 words, for program or data memory)
- ❑ DARAM B1 (256 words, for data memory)
- ❑ DARAM B2 (32 words, for data memory)

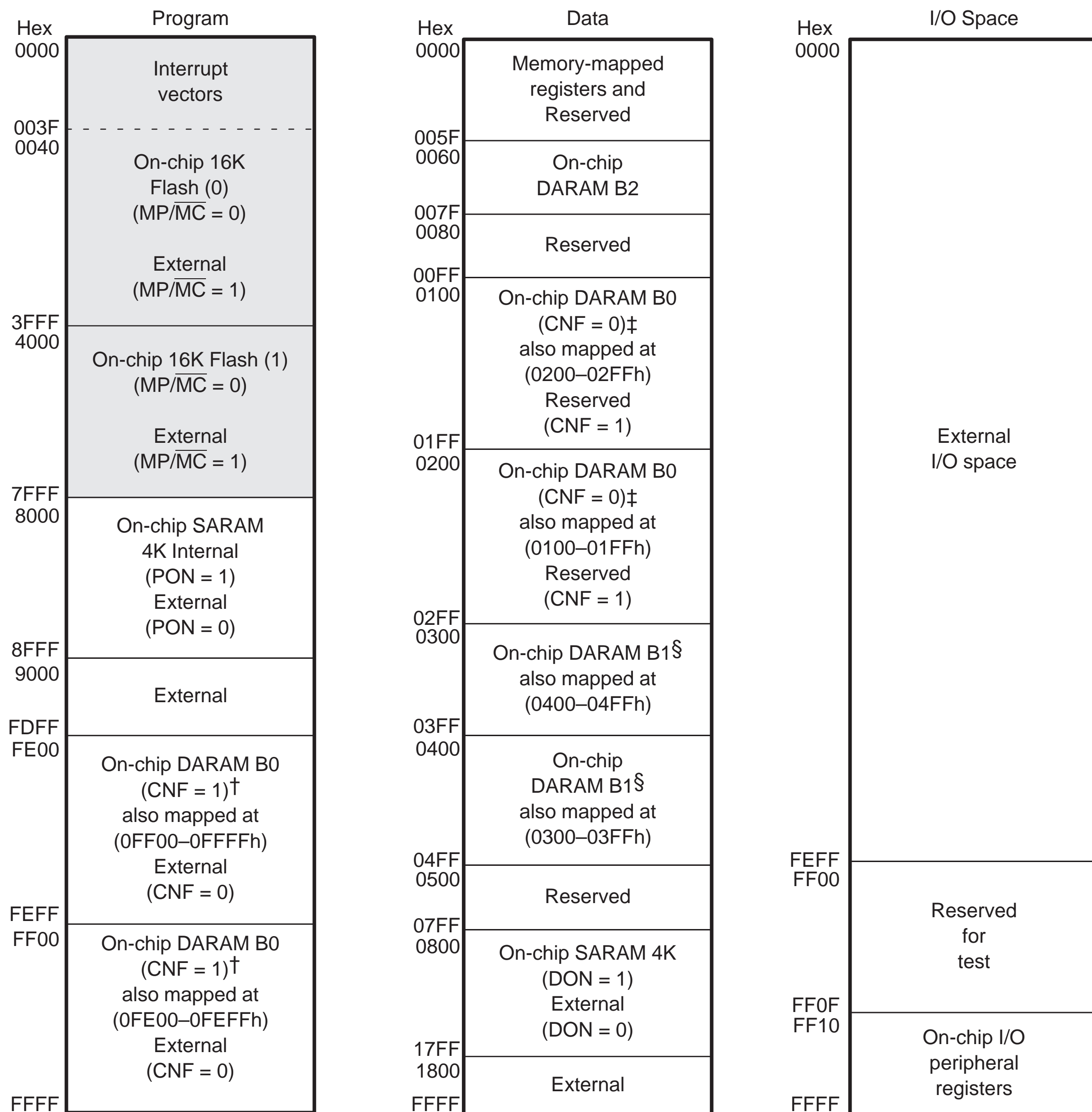
The ’F206 includes 544 x 16 words of dual-access RAM (DARAM), 4K x 16 single-access RAM (SARAM), and 32K x 16 program flash EEPROM memory. The PON and DON bits select the SARAM (4K) mapping in program, data or both. At reset, these bits are 11, mapping the SARAM in both program and data memory.

At reset, if the MP/$\overline{\text{MC}}$ is held high, the device is in microprocessor mode and the program address branches to 0000h (external program space). The MP/$\overline{\text{MC}}$ pin status is latched in the PMST register (bit 0). As long as this bit remains high, the device is in microprocessor mode. PMST register bits can be read and modified in software. If bit 0 is written 0, the device enters microcomputer mode and transfers control to the on-chip flash memory (0000h–7FFFh).

4.7.4 Flash Memory (EEPROM)

Flash EEPROM provides an attractive alternative to masked ROM. Like ROM, flash memory is non-volatile but has the added benefit of being electrically erasable and programmable without having to be removed from the target system. This “in-target” reprogrammability makes flash devices an attractive choice in the areas of prototyping, early field-testing and single-chip applications. Other key features of the flash include zero wait-state access and single 5-V power supply. The ’F206 incorporates two 16K x 16-bit flash EEPROM modules which provide a contiguous 32K x 16-bit array in program space. For further details on flash memory and programming, refer to the flash technical reference, *TMS320F20x/F24x DSP Embedded Flash Memory Technical Reference* (literature number SPRU282).

Figure 4–13. TMS320F206 Memory Map Configuration



† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to here as reserved when CNF = 1.

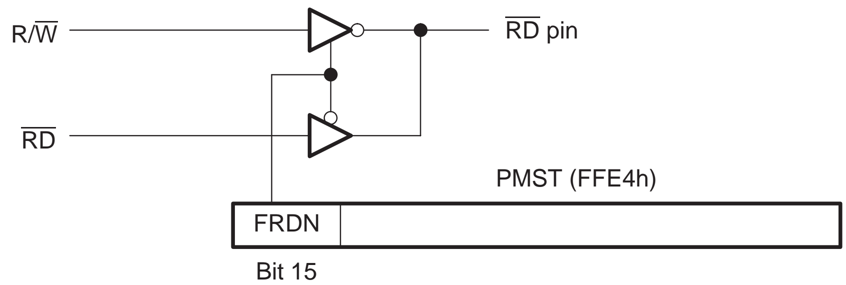
‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to here as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to here as reserved.

4.7.5 PMST Register in the ’206 Family

The PMST register provides improved memory interface options. This feature is in ’F206/LC206/C206 devices only. All the ’C20x DSP devices have critical external memory interface timings. At higher clock speeds, the existing $\overline{\text{RD}}$ signal is too delayed to be used as output enable for memory devices. In order to achieve a glueless zero wait state memory interface, $\overline{\text{RD}}$ signal has been provided with a software control bit. This bit (bit 15, FRDN) in PMST register (FFE4h) can select R/$\overline{\text{W}}$ as the new read signal (pin 45) instead of $\overline{\text{RD}}$ signal. Choosing R/$\overline{\text{W}}$ is necessary only if RD is incapable of supporting a zero wait state memory interface.

Figure 4–14. PMST Register Selection for $\overline{\text{RD}}$



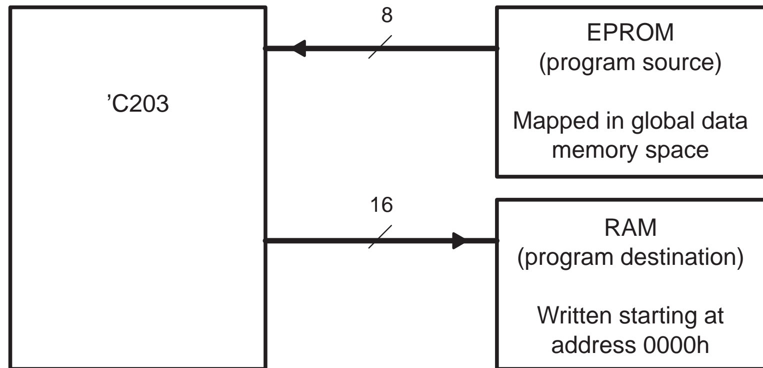
Notes: 1) RD is enabled at reset.

2) R/$\overline{\text{W}}$ is the RD pin signal for fast memory interface if FRDN is enabled .

4.8 'C203 Bootloader

This section applies to the 'C203's on-chip bootloader, which boots software from an 8-bit external ROM to a 16-bit external RAM at reset (see Figure 4–15). The source for your program is an external ROM located in external global data memory. The destination for the boot-loaded program is RAM in the program space. The main purpose of the bootloader is to provide you with the ability to use low-cost, simple-to-use 8-bit EPROMs with the 16-bit 'C20x.

Figure 4–15. Simplified Block Diagram of Bootloader Operation



The code for the bootloader is stored on chip. Using the bootloader requires several steps: choosing an EPROM, connecting and programming the EPROM, enabling the bootloader program, and finally, booting.

4.8.1 Choosing an EPROM

The code that you want boot-loaded must be stored in non-volatile external memory; usually, this code is stored in an EPROM. Most standard EPROMs can be used. At reset, the processor defaults to the maximum number of software wait states to accommodate slow EPROMs.

The maximum size for the EPROM is 32K words × 8 bits, which accommodates a program of up to 16K words × 16 bits. However, you could use the bootloader to load your own boot software to get around this limit or to perform a different type of boot.

Recommended EPROMs include the 27C32, 27C64, 27C128, and 27C256.

4.8.2 Connecting the EPROM to the Processor

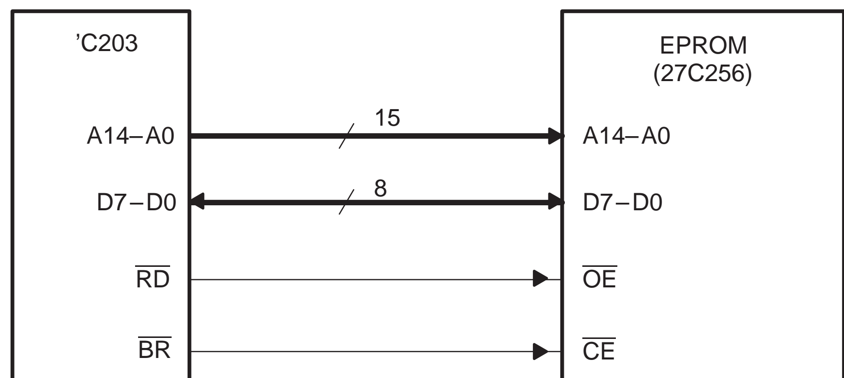
To map the EPROM into the global data space at address 8000h, make the following connections between the processor and the EPROM (refer to Figure 4–16):

- ❑ Connect the address lines of the processor and the EPROM (see lines A14–A0 in the figure).
- ❑ Connect the data lines of the processor and the EPROM (see lines D7–D0 in the figure).
- ❑ Connect the processor's $\overline{RD}$ pin to the EPROM output enable pin ($\overline{OE}$ in the figure).
- ❑ Connect the processor's $\overline{BR}$ pin to the EPROM chip enable pin ($\overline{CE}$ in the figure).

Notes:

1) If the EPROM is smaller than 32K words × 8 bits, connect only the address pins that are available on the EPROM.
2) When the bootloader accesses global memory, along with $\overline{BR}$, $\overline{DS}$ is driven low. Design your system so that the $\overline{DS}$ signal does not initiate undesired accesses to data memory during the boot loads.

Figure 4–16. Connecting the EPROM to the Processor



4.8.3 Programming the EPROM

Texas Instruments fixed-point development tools provide the utilities to generate the boot ROM code. The on-chip boot ROM is located at address FF00h and it is only accessible by the CPU during the boot-load process. After boot loading is complete, the boot ROM is removed from the memory map. (For an introduction to the procedure for generating bootloader code, see Appendix D, *Program Examples*.) However, should you need to do the programming, use the following procedure.

Store the following to the EPROM:

- ❑ Destination address. Store the destination address in the first two bytes of the EPROM—store the high-order byte of the destination address at EPROM address 8000h and store the low-order byte at EPROM address 8001h.

- ❑ Program length. Store N (the length of your program in words) in the next two bytes in EPROM. Use this calculation to determine N:

  N = ((number of bytes to be transferred)/2) – 1

  Store the high-order N byte at EPROM address 8002h and the low-order N byte at EPROM address 8003h.

- ❑ Program. Store the program, one byte at a time, beginning at EPROM address 8004h.

  Each word in the program must be divided into two bytes in the EPROM; store the high-order byte first and store the low-order byte second. For example, if the first word is 813Fh, you would store 81h into the first byte (at 8004h) and 3Fh into the second byte (at 8005h). Then, you would store the high byte of the next word at address 8006h.

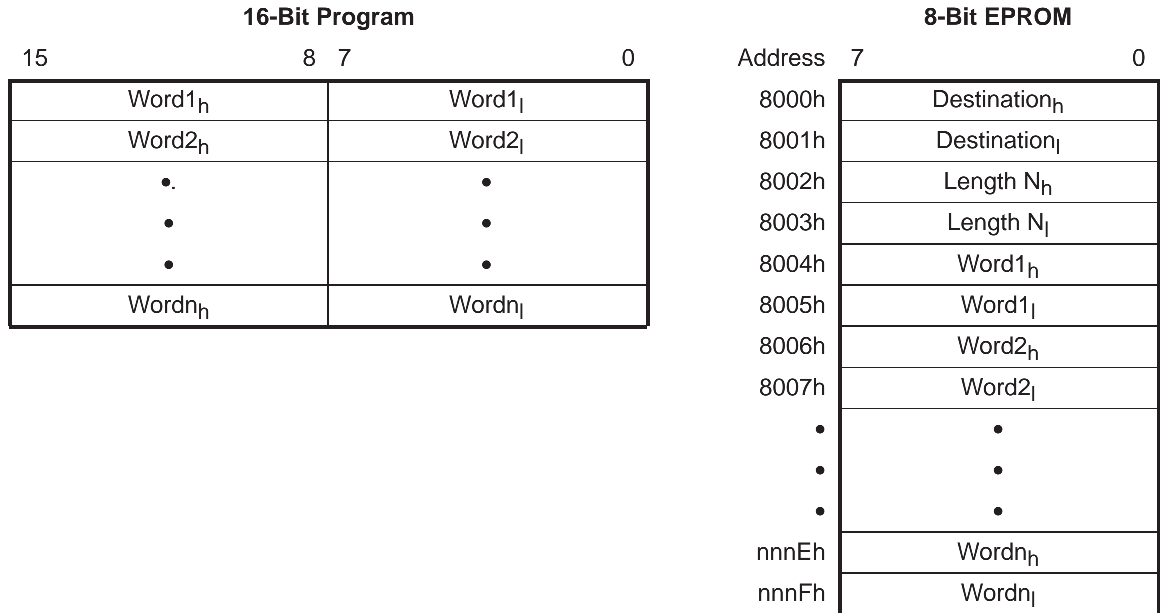
Notes:

1) Do not include the first four bytes of the EPROM in your calculation of the length (N). The bootloader uses N beginning at the fifth byte of the EPROM.

2) Make sure the first part of the program on the EPROM contains code for the reset and interrupt vectors. These vectors must be stored in the destination RAM first, so that they can be fetched from program-memory addresses 0000h–003Fh. The reset vector will be fetched from 0000h. For a list of all the assigned vector locations, see section 5.6.2, *Interrupt Table*, on page 5-16.

Figure 4–17 shows how to store a 16-bit program into the 8-bit EPROM. A subscript h (for example, on Word1$_h$) indicates the high-byte and a subscript l (for example, on Word1$_l$) indicates the low byte.

Figure 4–17. Storing the Program in the EPROM



4.8.4 Enabling the Bootloader

To enable the bootloader, tie the $\overline{\text{BOOT}}$ pin low and reset the device. The $\overline{\text{BOOT}}$ pin is sampled only at reset. If you do not want to use the bootloader, tie $\overline{\text{BOOT}}$ high before initiating a reset.

Three main conditions occur at reset that ensure proper operation of the bootloader:

- All maskable interrupts are globally disabled (INTM bit = 1).
- On-chip DARAM block B0 is mapped to data space (CNF bit = 0).
- Seven wait states are selected for program and data spaces.

After a hardware reset, the processor either executes the bootloader software or skips execution of the bootloader, depending on the level on the $\overline{\text{BOOT}}$ pin:

- If $\overline{\text{BOOT}}$ is low, the processor branches to the location of the on-chip bootloader program (FF00h).
- If $\overline{\text{BOOT}}$ is high, the processor begins program execution at the address pointed to by the reset vector at address 0000h in program memory.

4.8.5 Bootloader Execution

Once the EPROM has been programmed and installed, and the bootloader has been enabled, the processor automatically boots the program from EPROM at startup. If you need to reboot the processor during operation, bring the $\overline{RS}$ pin low to cause a hardware reset.

When the processor executes the bootloader, the program first enables the full 32K words of global data memory by setting the eight LSBs of the GREG register to 80h. Next, the bootloader copies your program from the EPROM in global data space to the RAM in program space through a five step process (refer to Figure 4–18):

1) The bootloader loads the first two bytes from the EPROM and uses this word as the destination address for the code. (In Figure 4–18, the destination is 0000h.)

2) The bootloader loads the next two bytes to determine the length of the code.

3) The bootloader transfers the next two bytes. It loads the high byte first and the low byte second, combines the two bytes into one word, stores the new word in the destination memory location, and then causes an increment in the source and destination addresses.

4) The bootloader checks to see if the end of the program has been reached:

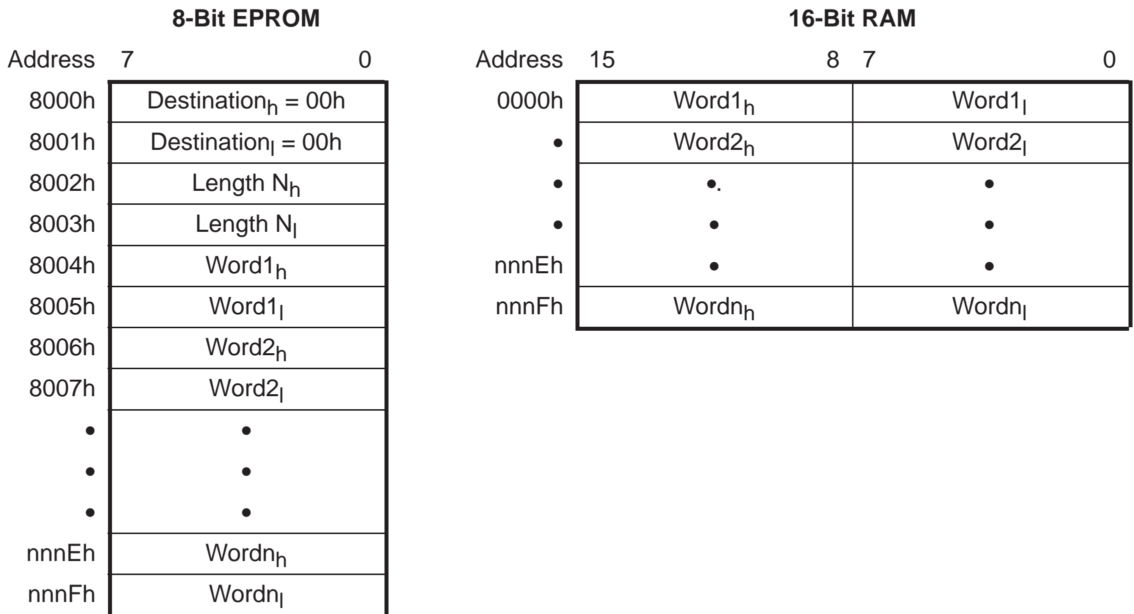
   - If the end is reached, the bootloader goes on to step 5.
   - If the end is not reached, the bootloader repeats steps 3 and 4.

5) The bootloader disables the entire global memory and then forces a branch to the reset vector at address 0000h in program memory. Once the bootloader finishes operation, the processor switches the on-chip bootloader out of the memory map.

Note:

During the boot load, data is read using the low-order eight data lines (D7–D0). The upper eight data lines are not used by the bootloader code.

Figure 4–18. Program Code Transferred From 8-Bit EPROM to 16-Bit RAM

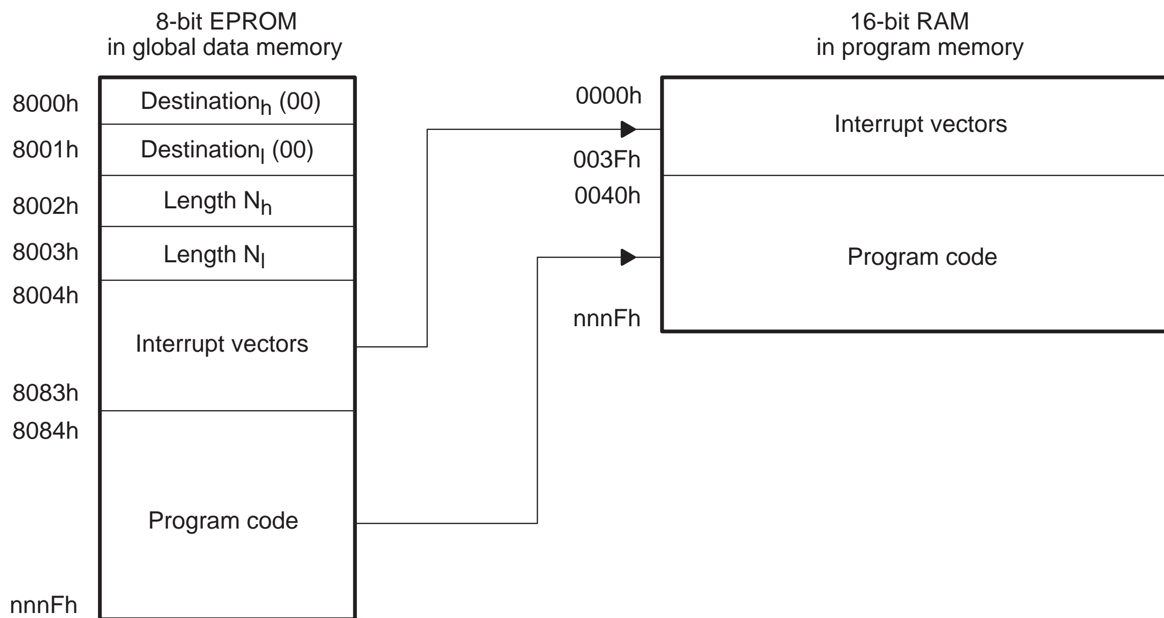


The ’C203 fetches its interrupt vectors from program-memory locations 0000h–003Fh (the reset vector is fetched from 0000h). Make sure that the interrupt vectors are stored at the top of the EPROM, so that they will be transferred to addresses 0000h–003Fh in the RAM (see Figure 4–19). Each interrupt vector is a branch instruction, which requires four 8-bit words, and there is space for 32 interrupt vectors. Therefore, the first 128 words to be transferred from the EPROM should be the interrupt vectors.

Note:

In the ’C203, the on-chip boot ROM is located at program address FF00h. It is accessed by the CPU only during the bootload process. After bootloading is complete, the boot ROM is removed from the memory map.

Figure 4–19. Interrupt Vectors Transferred First During Boot Load



4.8.6 Bootloader Program

```
********************************************************************************
*   TMS320C20x Bootloader Program                                              *
*                                                                              *
*   This code sets up and executes bootloader code that loads program          *
*   code from location 8000h in external global data space and transfers it   *
*   to the destination address specified by the first word read from locations *
*   8000h and 8001h.                                                           *
********************************************************************************
         .length  60
GREG     .set     5h             ; The GREG Register
SRC      .set     8000h          ; Source address
DEST     .set     60h            ; Destination address
LENGTH   .set     61h            ; Code length
TEMP     .set     62h            ; Temporary storage
HBYTE    .set     63h            ; Temporary storage for upper half of 16–bit word
CODEWORD .set     64h            ; Hold program code word
         .sect    ”bootload”
*
*  Initialization
*
BOOT     LDP      #0             ; Set the data page to 0 (load DP with 0)
         SPLK     #2E00h,TEMP    ; Set ARP = 1, OVM = 1, INTM = 1, DP = 0
         LST      #0,TEMP
         SPLK     #21FCh,TEMP    ; Set ARB = 1, CNF = 0, SXM = 0, XF = 1, PM = 0
         LST      #1,TEMP
         SPLK     #80h,GREG      ; Designate locations 8000–FFFFH as global data
                                 ; space
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       BOOT LOAD FROM 8–BIT MEMORY. MOST SIGNIFICANT BYTE IS FIRST       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
*  Determine destination address
*
ADDR     LAR      AR1,#SRC       ; AR1 points to global address 8000h
         LACC     *+,8           ; Load ACC with high byte shifted left by 8 bits
         SACL     HBYTE          ; Store high byte
         LACL     *+             ; Load ACC with low byte of destination
         AND      #0FFH          ; Mask off upper 24 bits.
         OR       HBYTE          ; OR ACC with high byte to form 16-bit
                                 ; destination address
         SACL     DEST           ; Store destination address
*
*  Determine length of code to be transferred
*
LEN      LACC     *+,8           ; Load ACC with high byte shifted left by 8 bits
         SACL     HBYTE          ; Store high byte
         LACL     *+             ; Load ACC with low byte of length
         AND      #0FFH          ; Mask off upper 24 bits.
         OR       HBYTE          ; OR ACC with high byte to form 16-bit length
         SACL     LENGTH         ; Store length
         LAR      AR0,LENGTH     ; Load AR0 with length to be used for BANZ
```

```
*
*  Transfer code
*
LOOP     LACC     *+,8          ; Load ACC with high byte of code shifted by 8 bits
         SACL     HBYTE         ; Store high byte
         LACL     *+,AR0        ; Load ACC with low byte of code
         AND      #0FFH         ; Mask off upper 24 bits
         OR       HBYTE         ; OR ACC with high byte to form 16-bit code word
         SACL     CODEWORD      ; Store code word
         LACL     DEST          ; Load destination address
         TBLW     CODEWORD      ; Transfer code to destination address
         ADD      #1            ; Add 1 to destination address
         SACL     DEST          ; Save new address
         BANZ     LOOP,AR1      ; Determine if end of code is reached
         SPLK     #0,GREG       ; Disable entire global memory
         INTR     0             ; Branch to reset vector and execute code.

         .END
```

Note:

The INTR instruction in the bootloader program causes the processor to push a return address onto the stack, but the device does not use a RET to return to this address. Therefore, your program must execute a POP instruction to get the address off the stack.

4.9 ’C206/LC206 Bootloader

This section describes the bootloader options available on the TMS320C206 and TMS320LC206. Several boot-load options are available on these devices. You can choose the option required by external pin configurations and an 8-bit word input from I/O address 0000h. The bootloader provides the flexibility of loading any executable code into the program memory of the DSP. Your code can be transferred to the DSP program memory from any one of the following external sources:

- 8/16-bit transfer through the synchronous serial port (SSP)
- 8-bit transfer through the asynchronous synchronous serial port (ASP)
- 8/16-bit EPROM
- 8/16-bit parallel port mapped to I/O space address 0001h of the DSP

Additionally, a warm boot is also supported.

4.9.1 Boot-load Options

The main function of the bootloader is to transfer user code from an external source to the program memory at power-up. The TMX320C206/LC206 provides several ways to download code to accommodate varying system requirements. To ensure compatibility, the ’C206 bootloader supports the original ‘C203 boot-load mode. The *EXT8* pin (pin 1) of the ’C206/’LC206 is sampled during startup to determine whether to perform the ‘C203 boot-load or the enhanced 206 boot-load options. are to be performed. Unlike the ‘C203 bootloader, the ’C206 bootloader can load multiple sections of user code in different segments of memory. In all boot-load modes, the processor automatically branches to the beginning your code, once boot loading is complete.

There are two possible scenarios for the TMS320C206/LC206 during startup based on the condition of the *EXT8* pin:

- EXT8 = low: This invokes the original ‘C203 style bootloader, which boot loads from an external 8-bit EPROM.
- EXT8 = high: This invokes the enhanced ’C206 bootloader, which supports the following boot-load options:
  - Synchronous serial port, 8/16 bit
  - UART/asynchronous serial port, 8 bit
  - External parallel EPROM, 8/16 bit
  - Parallel I/O boot, 8/16 bit using $\overline{\text{BIO}}$ and XF for handshaking
  - Warm boot

The option to be executed is determined by reading the word at I/O address 0000h. The lower 8-bits of the word specify which bootloader option to use.

4.9.2 Bootloader Operation

If the MP/$\overline{MC}$ pin is sampled low during a hardware reset, execution begins at location 0000h of the on-chip ROM. This location contains a branch instruction to the start of the bootloader program. The level of the EXT8 pin is read via bit 3 (LEVEXT8) in the PMST register (FFE4h in I/O space). If EXT8 pin is read high, the bootloader checks the boot selection word at location 0000h in I/O space and determines which booting method to execute. If EXT8 pin is read low, control passes by default to 8-bit EPROM boot (‘C203 style bootloader). This allows upward compatibility from TMS320C203. Figure 4–20 shows the PMST register. Table 4–7 describes the function of the PMST register bits. Table 4–8 shows bootloader pin configuration.

Figure 4–20. Program Memory Status (PMST) Register – (I/O space FFE4h)

15	14–4	3	2	1	0
FRDN	Reserved	LEVEXT8	DON	PON	MP/$\overline{MC}$
R/W	0	R	R/W	R/W	R/W

Table 4–7. PMST Register Bit Descriptions

Bit	Name	Value at Reset	Function
15	FRDN	0	At reset, this bit is 0, which enables enhanced $\overline{RD}$ signal. If high, the inverted R/$\overline{W}$ is active.
14–4	Reserved	0	These bits are not used.
3	LEVEXT8	x	Bit 3 (a read-only bit) latches in the state of EXT8 pin at reset. If low, the on-chip bootloader uses ‘C203 style boot load. If high, the enhanced ’C206 bootloader is used.
2	DON	1	See below.
1	PON	1	Bit 1 and bit 2 configure the SARAM mapping either in program memory, data memory or both. At reset these bits are 11. DON (bit 2) PON (bit 1) 0 0 SARAM not mapped, address in external memory 0 1 SARAM in program memory at 0x8000h 1 0 SARAM in data memory at 0x800h 1 1 SARAM in program and data memory (reset value)
0	MP/$\overline{MC}$	x	Bit 0 latches in the state of MP/$\overline{MC}$ at reset. This bit can also be written to switch between Microprocessor (1) or Microcomputer (0) modes.

Table 4–8. Bootloader-Pin Configuration

MP/$\overline{MC}$	EXT8	Option	Mode(s)
0	0	Use ‘C203 style bootloader	1
0	1	Use ’C206 enhanced bootloader	2 to 9
1	0	EXT8 has no effect	–
1	1	EXT8 has no effect	–

The bootloader sets up the CPU status registers as follows:

- ❑ On-chip DARAM block B0 is mapped into program space (CNF = 1).
- ❑ On-chip SARAM block is mapped into program and data space (PON = 1, DON=1).

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.

At reset, interrupts are globally disabled (INTM = 1). Entire program and data memory spaces are enabled with seven wait states.

4.9.3 ’C206 Enhanced Bootloader (EXT8 High - Modes 2 to 9)

The bootloader reads the I/O port address 0000h by driving the I/O strobe ($\overline{IS}$) signal low. The lower eight bits of the word read from I/O port address 0000h specify the mode of transfer; the higher eight bits are ignored. This boot-routine-selection (BRS) word determines the boot mode. The BRS word uses a 6-bit source address field (SRCE_AD) in parallel EPROM mode and a 6-bit entry address field (ADDR_bb) in warm-boot mode to arrive at the starting address of the code.

Figure 4–21 lists the available boot-load options and the corresponding values for the boot-routine-selection word at I/O address 0000h. This word could be set by a DIP switch.

Figure 4–22 shows the available boot-load options in flow chart form.

Figure 4–21. Enhanced ’C206 Bootloader Options

BRS word @ I/O 0000h			Boot Load Option	Mode
xxxxxxxx	xxx0	0000	8-bit serial SSP, external FSX, CLKX	2
xxxxxxxx	xxx0	0100	16-bit serial SSP, external FSX,CLKX	3
xxxxxxxx	xxx0	1000	8-bit parallel I/O	4
xxxxxxxx	xxx0	1100	16-bit parallel I/O	5
xxxxxxxx	xxx1	0000	8-bit ASP /UART	6
xxxxxxxx	SRCE	AD01	8-bit EPROM	7
xxxxxxxx	SRCE	AD10	16-bit EPROM	8
xxxxxxxx	ADDR	bb11	Warm-boot	9

Figure 4–22. Boot-load Flowchart

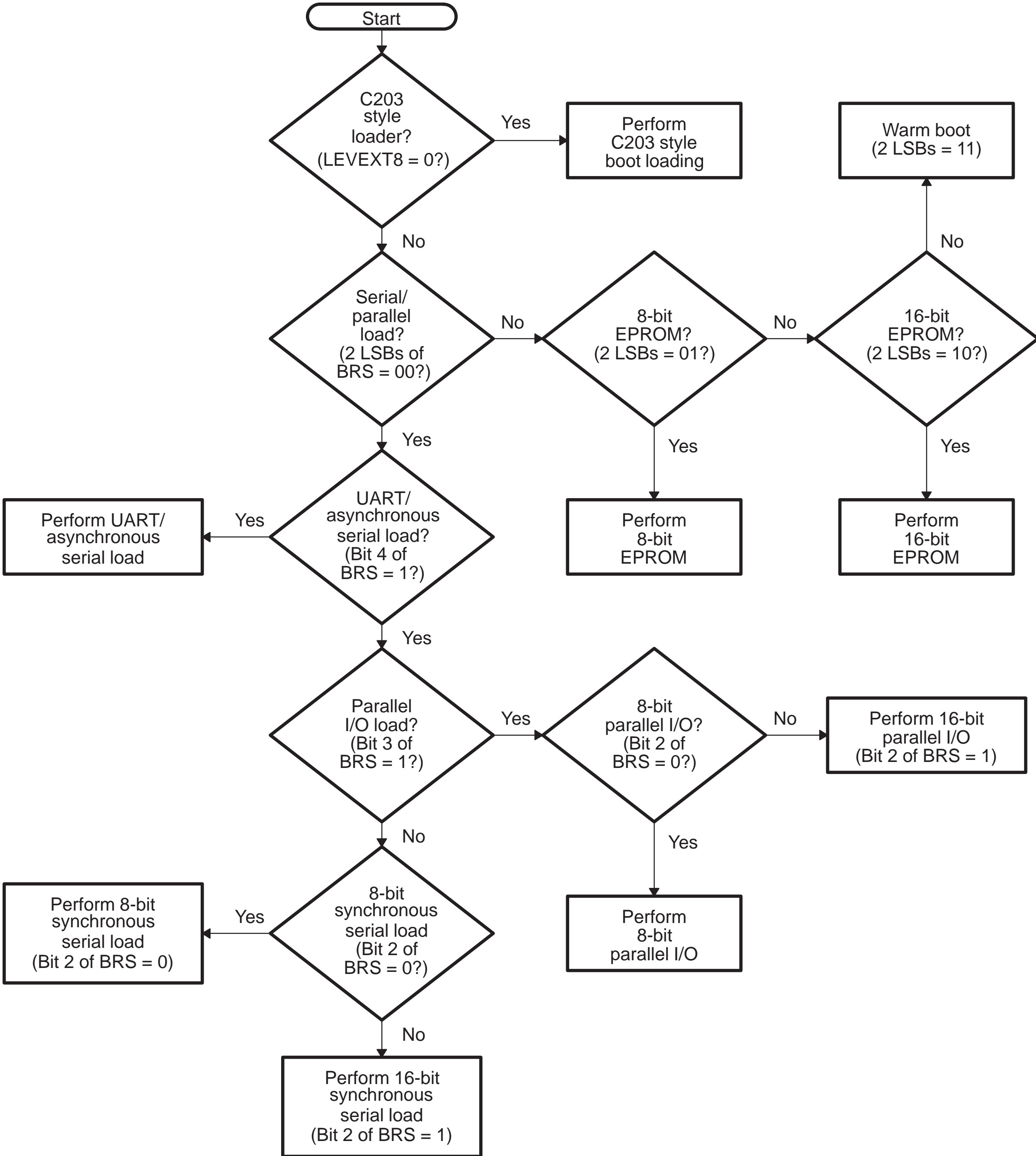
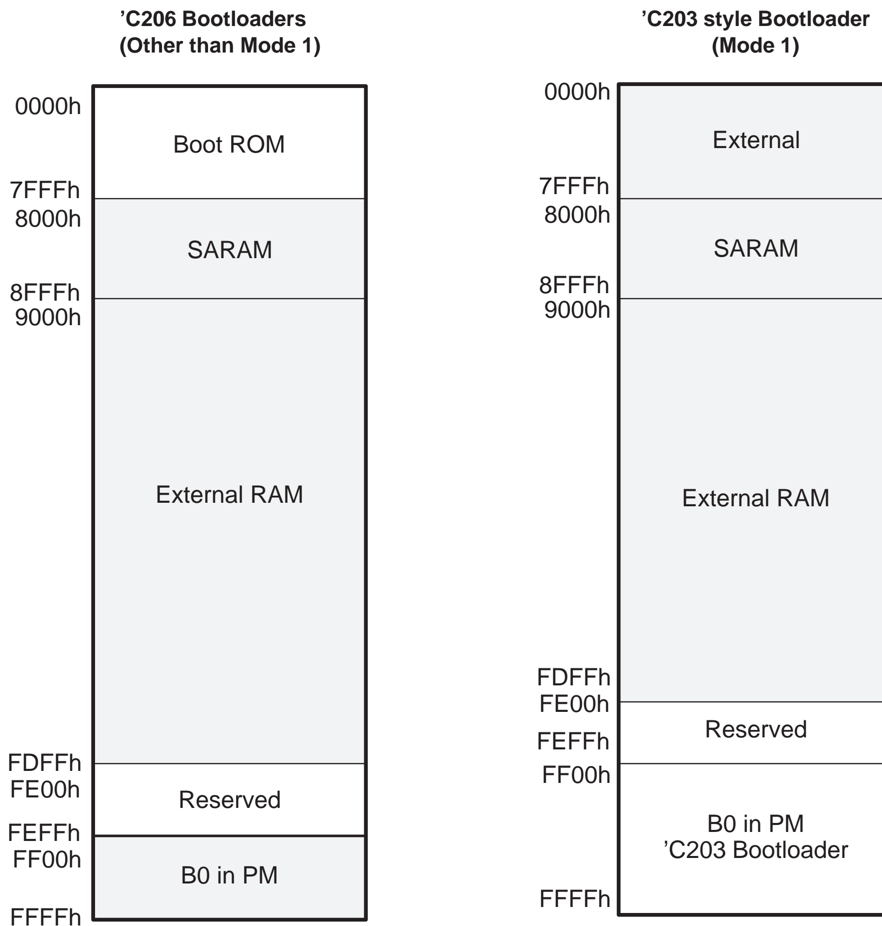


Figure 4–23 provides the memory map of program address spaces that are accessible through the bootloader. For modes other than 1, memory locations from 0000h to 7FFFh are not available for loading code, since that space is occupied by ROM. However, this limitation can be overcome by modifying the memory map in your own boot code.

Figure 4–23. Destination Address Space for Programs in Program Memory



Caution: Locations 8000h - 807Fh in SARAM are reserved for the second interrupt vector table as mentioned in section 5. Exercise caution while moving code into this area.

4.9.4 Interrupt Vectoring

Interrupt vectors stored in the on-chip ROM have hard coded addresses to the on-chip SARAM starting at address 8000h in program space. When an interrupt occurs, a branch is made to the corresponding interrupt vector located in the on-chip ROM at addresses (0000h–0040h). A branch instruction then transfers program control to the second interrupt vector table in the on-chip SARAM. You must initialize the second interrupt vector table. This table is used to allow remappable interrupt vectors. See the following code for initializing interrupt vectors in the SARAM.

Remapped interrupt vectors for TMS320C206, TMS320LC206

```
int1_holdv          .set   8000h  ; User maskable interrupt #1
int2_3v             .set   8002h  ; User maskable interrupts #2 & #3
tintv               .set   8004h  ; Timer interrupt vector
rintv               .set   801Ah  ; SSP receive interrupt vector
xintv               .set   8032h  ; SSP transmit interrupt vector
txrxintv            .set   804Eh  ; UART port Tx/Rx interrupt vector
trapv               .set   8050h  ; Software trap vector
nmiv                .set   8052h  ; Non-maskable interrupt vector
swi8v               .set   8054h  ; Software interrupt vectors begin...
swi9v               .set   8056h
swi10v              .set   8058h  ; (Note:If these interrupts are unused
swi11v              .set   805Ah  ; these memory locations may be
swi12v              .set   805Ch  ; used for other purposes.)
swi13v              .set   805Eh
Si14v               .set   8060h
swi15v              .set   8062h
swi16v              .set   8064h
swi20v              .set   8066h
swi21v              .set   8068h
swi22v              .set   806Ah
swi23v              .set   806Ch
swi24v              .set   806Eh
swi25v              .set   8070h
swi26v              .set   8072h
swi27v              .set   8074h
swi28v              .set   8076h
swi29v              .set   8078h
swi30v              .set   807Ah
swi31v              .set   807Ch
reserved            .set   807Eh
```

4.9.5 Synchronous Serial Port (SSP) Boot Mode

In this mode, the synchronous serial port control register (SSPCR) is configured for 16-bit or 8-bit word transfer. The data shift clock and frame sync must be supplied by the external device to the ’C206/’LC206.

- 16-Bit Word Serial Transfer (Mode 3)

  If the 16-bit word transfer is selected , the first 16-bit word received by the ’C206 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 ’C206 checks to see if there are any more sections to be transferred. If there are additional sections to be transferred, the bootloader proceeds to transfer them in exactly the same way as the first section. After transferring all the sections, the ’C206 branches to the first destination address. The length N is defined as:

  N = (Number of 16-bit words) - 1

  If, after transferring all the N words of a section, the ’C206 receives a 0000, it signals the end of user code. If any word other than 0000 is read, it indicates that one or more sections is following and the word read is treated as the destination address of the next section. Refer to Figure 4–24 for the format of data transfer in 16-bit mode.

Figure 4–24. 16-Bit Word Transfer

$DESTINATION_1$
LENGTH of first section (N_1)
CODE(1) of length N_1
$DESTINATION_2$
LENGTH of second section (N_2)
CODE(2) of length N_2
$DESTINATION_N$
LENGTH of Nth section (N_N)
CODE(N) of length N_N
0000 to end program

Legend:

$Destination_{16}$	16-bit destination address
$Length_{16}$	16-bit word that specifies the length of the code (N) that follows
$Code(N)_{16}$	N number of 16-bit words to be transferred (actual code)

- ❑ 8-Bit Word Serial Transfer (Mode 2)

  If the 8-bit word transfer is selected , a higher-order byte and a lower-order byte form a 16-bit word. The first 16-bits received by the ’C206 from the serial port specify the destination address ($Destination_h$ and $Destination_l$) of code in program memory. The next 16-bits specify 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 ’C206 checks to see if there are any more sections to be transferred. If there are additional sections to be transferred, the bootloader proceeds to transfer them in exactly the same way as the first section. After transferring all the sections, the ’C206 branches to the first destination address. The length N is defined as:

  N = (Number of 16-bit words) - 1

  or

N = (Number of bytes to be transferred/2) - 1

If, after transferring all the N words of a section, the 'C206 receives a 0000, it signals the end of user code. If any word other than 0000 is read, it indicates that one or more sections is following and the word read is treated as the destination address of the next section. Refer to Figure 4–26 for the format of data transfer in 8-bit mode. Figure 4–25 shows the connection details for SSP boot-load option.

Figure 4–25. Host-'C206 Interface for SSP Boot-load Option

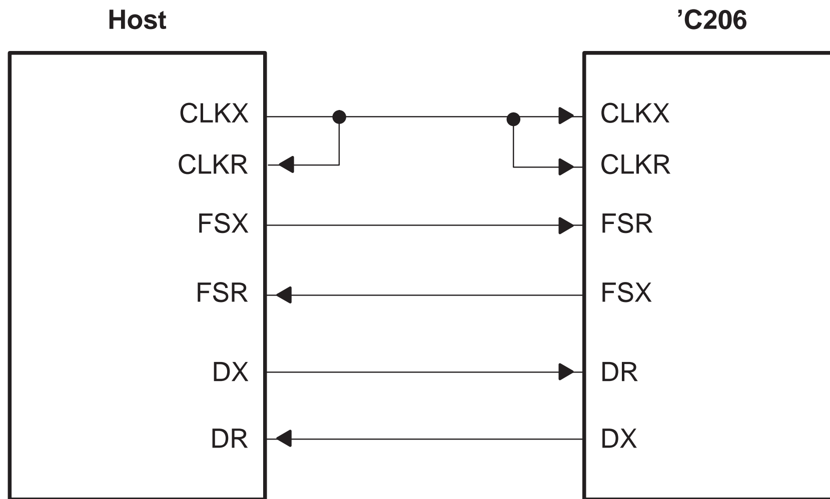


Figure 4–26. Figure 9. 8-Bit Word Transfer

$DESTINATION_{1h}$
$DESTINATION_{1l}$
$LENGTH_h$ of first section (N_{1h})
$LENGTH_l$ of first section (N_{1l})
$CODE(1)_h$
$CODE(1)_l$
$DESTINATION_{2h}$
$DESTINATION_{2l}$
$LENGTH_h$ of second section (N_{2h})
$LENGTH_l$ of second section (N_{2l})
$CODE(2)_h$
$CODE(2)_l$
$DESTINATION_{Nh}$
$DESTINATION_{Nl}$
$LENGTH_h$ of Nth section
$LENGTH_l$ of Nth section
$CODE(N)_h$
$CODE(N)_l$
0000 to end program

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 $(N)_h$	High byte of N number of 16-bit words to be transferred
Code $(N)_l$	Low byte of N number of 16-bit words to be transferred

4.9.6 UART/Asynchronous Serial Port (ASP) Boot Mode (Mode 6)

This mode is extremely useful to transfer user code to the ’206 through an asynchronous serial port such as the RS-232 port available in personal computers. The data packet format in this mode is similar to that of synchronous serial port (SSP) boot mode, with the exception that only 8-bit transfers are supported. The DSPHEX utility is used to convert the COFF file (*.out) of the user to a hex file suitable for UART bootloading. For more information about the DSPHEX utility, refer to *TMS320C1x/C2x/C20x/C5x Assembly Language Tools User’s Guide* (literature number SPRU018D).

The ’206 senses the baud rate of the incoming data and automatically updates its baud-rate register. To make this happen, the host must transmit the ASCII character “a” (or “A”) in the very beginning of the data transfer. ’C206 boot code echoes “a” on baud lock and then prepares itself to receive user code. The DSPHEX utility does not automatically add the ASCII value of the character “a” in the hex file it creates. You can do this with the help of any ASCII editor. While editing the hex file, you must also make sure that the last word of the file is 0000h in order to transfer control to the user code after boot loading. The options for the DSPHEX utility can be either specified on the command line or with the help of a command file. A sample command file for the DSPHEX utility is given below:

```
/* DSPHEX command file to generate hex file from .out file          */
/* suitable for UART bootloader                                     */

usercode.out        /* Replace with the actual name of user code */
–a                  /* ASCII- hex format                         */
–o usercode.hex     /* Replace with the reqd. name of user code  */
-byte               /* default                                   */
–order MS           /* default                                   */
–memwidth 8
–romwidth 8

SECTIONS
{ .text   : boot }
```

4.9.7 Parallel EPROM Boot Mode

The parallel EPROM boot mode is used when code is stored in EPROMs (8-bit or 16-bit wide). The code is transferred from external 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 SRCE_AD field of the boot routine selection word. A 16-bit source address is formed with the help of this SRCE_AD field as shown in Figure 4–27. The boot-load code initializes the GREG register to external global data memory space 8000h–0FFFFh. The ’C206/’LC206 transfers control to the source address after disabling global data memory.

Figure 4–27. 16-Bit Source Address for Parallel EPROM Boot Mode

15 – 10	9									0
SRCE_AD	0	0	0	0	0	0	0	0	0	0
Source address										

Note: SRCE_AD = 6-bit page address

- 16-Bit EPROM Transfer (Mode 8)

  If the 16-bit mode is selected, boot code is read in 16-bit words starting at the source address. After every read operation, the source address changes by an increment of 1 . 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 ’C206 checks to see if there are any more sections to be transferred. If there are additional sections to be transferred, the bootloader proceeds to transfer them in exactly the same way as the first section. After transferring all the sections, the ’C206 branches to the first destination address. The length N is defined as:

  N = (Number of 16-bit words) - 1

  If, after transferring all the N words of a section, the ’C206 receives a 0000, it signals the end of code. If any word other than 0000 is read, it indicates that one or more sections is following and the word read is treated as the destination address of the next section. Refer to Figure 4–24 for the format of data transfer in 16-bit mode.

  Note: 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-Bit EPROM Transfer (Mode 7)

  If the 8-bit mode is selected, 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 ’C206 checks to see if there are any more sections to be transferred. If there are additional sections to be transferred, the bootloader proceeds to transfer them in exactly the same way as the first section. After transferring all the sections, the ’C206 branches to the first destination address. The length N is defined as:

N = (Number of 16-bit words) - 1
or
N = (Number of bytes to be transferred/2) - 1

If, after transferring all the N words of a section, the ’C206 receives a 0000, it signals the end of user code. If any word other than 0000 is read, it indicates that one or more sections is following and the word read is treated as the destination address of the next section. Refer to Figure 4–26 for the format of data transfer in 8-bit mode.

Note: 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.

4.9.8 Parallel I/O Boot Mode (Mode 4 - 8 Bit, Mode 5 - 16 Bit)

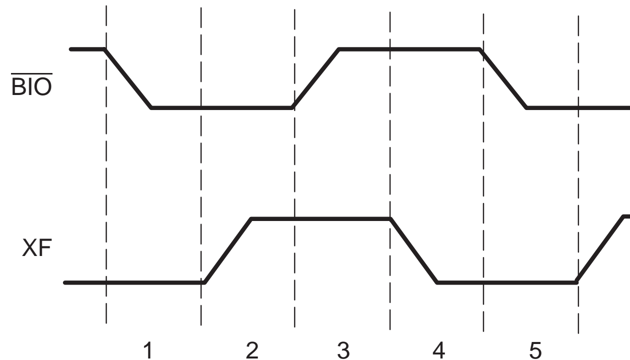
The parallel I/O boot mode asynchronously transfers code from I/O port at address 0001h to internal or external program memory. Each word can be 16 bits or 8 bits long and follows the same sequence outlined in parallel EPROM mode. The ’C206/’LC206 communicates with the external device using the BIO and XF lines for handshaking. This allows a slower host processor to communicate with the ’C206/’LC206 by polling/driving the XF and BIO lines. The handshake protocol shown in Figure 4–28 must be used to successfully transfer each word via I/O port 0001h.

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 0001h, ignoring the higher byte on the data bus.

A data transfer is initiated by the host, driving the BIO pin low. When the BIO pin goes low, the ’C206 inputs the data from I/O address 0001h, drives the XF pin high to indicate to the host that the data has been received and then writes the input data to the destination address. The ’C206 then waits for the BIO pin to go low before driving the XF pin low. The low status of the XF line can then be polled by the host for the next data transfer.

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 ’C206 accesses the external bus only when XF is high.

Figure 4–28. Handshake Protocol



Notes:
1) Host requests data transfer to ’C206 by making BIO low.
2) ’C206 reads in the data through I/O port 1 and makes XF high. Bootloader program loops until BIO becomes high.
3) After BIO is made high, bootloader acknowledges by making XF low indicating that it is ready for new data.
4) Bootloader program loops until BIO becomes low. XF continues to be low.
5) When BIO becomes low, it signals the host request for the transmission of the next word and the whole sequence repeats until all words are transferred.

4.9.9 Warm-Boot Mode (Mode 9)

The warm-boot operation does not move any code. It is useful to branch to your code if the code has already been transferred to internal or external program memory by other boot-load methods. This mode is used only if a “warm” device reset is required. Since warm-boot mode can be invoked only in the microcomputer mode, the first section of your code can reside only from 8000h onwards in program memory, as 0000h to 7FFFh is occupied by ROM. The six MSBs of the entry address are specified by the ADDR_bb field of the boot routine selection word (Figure 4–21). A 16-bit entry address is defined by this ADDR_bb field as shown in Figure 4–29. Since bits 0 – 9 are zero, the starting

address must lie on 400 word boundaries (x000h, x400h, x800h and xC00h). During initial boot load, the destination address of your code is stored in a memory variable in B2 RAM. The warm-boot routine uses this address to transfer control to the user code. If your application overwrites this memory location, then the address your code must be specified in the BRS word for warm-boot to function. The ’C206/’LC206 transfers control to the entry address after disabling global data memory.

Figure 4–29. 16-Bit Entry Address for Warm-Boot Mode

15–10	9	8	7	6	5	4	3	2	1	0
ADDR_bb	0	0	0	0	0	0	0	0	0	0
Entry address										

Note: ADDR_bb = 6-bit page address in 400h word boundaries

4.9.10 ’C203 Style Bootloader (EXT8 Low – Mode 1)

The ’C206 bootloader supports the ’C203 style bootloader when the EXT8 pin is tied low. However, there are some differences between the original ’C203 device bootloader and the ’C203 style bootloader option supported in the ’C206. This paragraph applies to the ’C203 device bootloader only. The bootloader option in the TMS320C203 device has a fixed destination address for the user code. This address must be 0000h, as the interrupt vector table must be modified first. The reset vector (0000h in program memory) must be initialized to point to the beginning of the user code. Other interrupt vectors may need to be setup depending on the user application. After the user code is boot loaded (for example, application code transferred to external program memory), *INTR 0* instruction is invoked by the bootloader. This transfers program control to your code. The boot source address (the address at which your code is stored in external non-volatile memory) is fixed at 8000h in global memory space.

The bootloader in TMX320C206/LC206 devices features an 8-bit boot option from external non-volatile memory (EPROM) to external SRAM or internal memory at reset if MP/$\overline{\text{MC}}$ pin is sampled low and EXT8 pin is tied low during a hardware reset. This mode is similar in operation to the original ’C203 device bootloader except during the final branch. There is no *INTR 0* instruction, rather program control branches to the address specified by the accumulator.

The maximum size of the EPROM can be 32K x 8 to yield 16K x 16 of program memory. However, you could boot your own bootloader, which would perform a function as desired. The bootloader begins loading from a fixed source address 8000h in external global data space and begins transferring to the

destination address in program space defined by you. This destination address is defined by the first two bytes of the EPROM. The destination address is not constrained to be 0000h as in the case of ’C203 device and can be any valid program address. However, you may need to modify the interrupt vector table.

At reset, interrupts are globally disabled, INTM = 1, B0 is mapped to program space, CNF = 1, and seven wait states are selected for program and data spaces. The boot-load code initializes the GREG register to external global data memory space 8000h–FFFFh. The operation of this mode is similar to 8-bit EPROM transfer (’C206 boot mode 7).

Note: The assembly source code for the ’C206 bootloader is available on the web at **www.ti.com** under ***’C20x DSPs***.

4.9.11 Bootloader Program

```
*
* TMS320C206/TMS320LC206 Bootloader Program
*   Revision 1.0, 12/18/97
*
*   Revision 1.2, 6/29/98
*
*   1.1 changes
*   1. Fix 16 bit EPROM load, need pointer for counter
*   2. Fix branching in serial I/O from EQ to TC
*   3. Change original 8 bit boot from using INTR 0 to a BACC instruction
*      and copy boot routine to B0.  This allows code to be copied to
*      address 0x0h after switching to microprocessor mode.
*   4. Set CNF = program space.
*   5. Add lacl in parallel 16 bit routine to load TEMP
*   6. Change TEMP to TEMP1 for 8 bit parallel I/O.
*
*
*   1.2 Changes
*   1. Change the branch address to 0xFF18 due to incorrect copy.
*   2. Changed address for DMOV on warm boot
*
*   Objective: This bootloader has a total of 9 options and is backward
*   compatible to the original '203 bootloader.
*
*   Operation:  Given the MP/MC pin is low at reset, the bootloader program
*               stored in the on-chip ROM determines which method of booting
                is to be used.
*               First, the program decides if the old method of 8 bit EPROM
*               boot is to be used.  If not it continues by reading I/O port
*               zero via the LEVEXT8 bit in the PMST register which is a direct
*               representation of pin 1 (EXT8).
*
*   Below are the options for reading I/O port 0:
*
*   16 BIT DATA BUS
*
*      8 bit SSP                  XXXX XXXX XXX0 0000
*      16 bit SSP                 XXXX XXXX XXX0 0100
*      8 bit parallel I/O         XXXX XXXX XXX0 1000
*      16 bit parallel I/O        XXXX XXXX XXX0 1100
*      ASP                        XXXX XXXX XXX1 0000
*      8 bit EPROM                XXXX XXXX SRC. ..01
*      16 bit EPROM               XXXX XXXX SRC. ..10
*      Warm boot                  XXXX XXXX ADR. ..11
*
*  Interrupt Vectoring:    Interrupt vectors stored in the on-chip ROM have hard
*                          coded addresses to the on-chip SARAM starting at
*                          address 0x8000 in program space.
*
*  Multiple sections booting:     The bootloader allows multiple sections of
*                                 program code to be copied via any of the options
*                                 except the old style '203 bootloader.
```

```
*                              The first section copied is assumed to be the
*                              entry point to the program once all section(s)
*                              have been copied.
*
* Note: B2PA_3 stores the address where execution begins from, after all
* sections have been loaded
**** Use C206BOOT.CMD file for linking *****
                .copy "sldrv201.h"        ; Variable and register declaration

*********************************************************************************
SRC         .set    8000h           ; source address
DEST        .set    60h             ; destination address
DEST1       .set    331h
LENGTH      .set    61h             ; code length
TEMP        .set    62h             ; temporary register
HBYTE       .set    63h             ; temporary storage for upper half of
                                    ; 16-bit word
TEMP1       .set    68h
CODEWORD    .set    64h             ; hold program code word
CODEWORD1   .set    330h            ; hold address for copy for oldboot routine
brs         .set    65h             ; Boot Selection Word
SOURCE      .set    66h
DEST2       .set    67h
b0          .set    0Fh
b1          .set    0Eh
b2          .set    0Dh
b3          .set    0Ch
b4          .set    0Bh

*  Interrupt vectors for TMS320C206, TMS320LC206
*
int1_holdv    .set   8000h        ; external interrupt vectors
int2_3v       .set   8002h        ;
tintv         .set   8004h        ; timer interrupt vector
rintv         .set   801Ah        ; receive interrupt vector
xintv         .set   8032h        ; transmit interrupt vector
txrxintv      .set   804Eh        ; UART port interrupt vector
trapv         .set   8050h        ; software trap vector
nmiv          .set   8052h        ; non-maskable interrupt vector
swi8v         .set   8054h        ; software interrupt vectors
swi9v         .set   8056h        ;
swi10v        .set   8058h        ; (Note: If these interrupts are unused
swi11v        .set   805Ah        ; these data memory locations can be
swi12v        .set   805Ch        ; assigned to other purposes.)
swi13v        .set   805Eh        ; Software interrupt vectors
swi14v        .set   8060h        ; |                    |
swi15v        .set   8062h        ; |                    |
swi16v        .set   8064h        ; V                    V
swi20v        .set   8066h        ;
swi21v        .set   8068h        ;
swi22v        .set   806Ah        ;
swi23v        .set   806Ch        ;
swi24v        .set   806Eh        ;
swi25v        .set   8070h        ;
```

```
swi26v        .set   8072h         ;
swi27v        .set   8074h         ;
swi28v        .set   8076h         ;
swi29v        .set   8078h         ;
swi30v        .set   807Ah         ;
swi31v        .set   807Ch         ;
reserved      .set   807Eh

*****************************************************************
       .sect   ”vectors”
*****************************************************************
reset  B      boot          ; 0  – power on reset
int1h  B      int1_holdv    ; 1  – external interrupt 1 or HOLD
int23  B      int2_3v             ; 2  – external interrupts 2 or 3
tint   B      tintv         ; 3  – timer interrupt
rint   B      rintv         ; 4  – synchronous serial port receive interrupt
xint   B      xintv         ; 5  – synchronous serial port transmit interrupt
txrx   B      txrxintv      ; 6  – asynchronous serial port transmit and
                            ;      receive interrupt
res    B      reserved      ; 7  – reserved for emulation
swi8   B      swi8v         ; 8  – software interrupt
swi9   B      swi9v         ; 9  – software interrupt
swi10  B      swi10v        ; 10 – software interrupt
swi11  B      swi11v        ; 11 – software interrupt
swi12  B      swi12v        ; 12 – software interrupt
swi13  B      swi13v        ; 13 – software interrupt
swi14  B      swi14v        ; 14 – software interrupt
swi15  B      swi15v        ; 15 – software interrupt
swi16  B      swi16v        ; 16 – software interrupt
trap   B      trapv         ; 17 – software trap
nmi    B      nmiv          ; 18 – non–maskable interrupt
res1   B      reserved      ; 19 – Reserved
swi20  B      swi20v        ; 20 – software interrupt
swi21  B      swi21v        ; 21 – software interrupt
swi22  B      swi22v        ; 22 – software interrupt
swi23  B      swi23v        ; 23 – software interrupt
swi24  B      swi24v        ; 24 – software interrupt
swi25  B      swi25v        ; 25 – software interrupt
swi26  B      swi26v        ; 26 – software interrupt
swi27  B      swi27v        ; 27 – software interrupt
swi28  B      swi28v        ; 28 – software interrupt
swi29  B      swi29v        ; 29 – software interrupt
swi30  B      swi30v        ; 30 – software interrupt
swi31  B      swi31v        ; 31 – software interrupt
       .sect   ”bootload”

*  Initialization
boot   LDP    #0
       SPLK   #2E00H,TEMP   ; ARP = 1, OVM = 1, INTM = 1, DP = 0
       LST    #0,TEMP       ; B0 is in PM
       SPLK   #31FCH,TEMP   ; ARB = 1, CNF = 1, SXM = 0
       LST    #1,TEMP       ; XF = 1, PM = 0 ,  B0-->Prog.memory

******************************************************
```

```
*  Determine if old or new boot method                *
******************************************************
       IN     TEMP,PMST     ; Read level of EXT8 pin.
       BIT    TEMP,b3       ; Test LEVEXT8 bit.
       BCND   OLDBOOT,NTC   ; Branch to 8-bit EPROM boot.
                            ; nextsect = 0    FDEST = 1
       splk   #0,nextsect   ; flag for determining if new section exists
       splk   #1,FDEST      ; FLAG to determine address of code entry
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*          Read Configuration Byte                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
       IN     brs,0h        ; read I/O port 0   (I/O 0 -->65h)
       LACC   brs,8         ; Shifted BRS word --> ACC
       AND    #0FC00h       ; throw away 2 LSBs
       SACL   SOURCE        ; save as source address
                            ; b15.....b10 b9 b8 0000 0000 -->SOURCE
       LACL   brs           ; BRS -->ACC
       AND    #3            ; if 2 LSBs == 00
       BCND   ser_io,eq     ; use serial or parallel I/O or ASP
                            ; At this stage, b1 b0 can be 01,10 or 11
       sub    #2            ; if 2 LSBs == 01
       bcnd   PAR08,lt      ;  load from 8-bit memory (EPROM)
                            ; if 2 LSBs == 10
       bcnd   PAR16,eq      ;  load from 16-bit memory (EPROM)
                            ; else 2 LSBs == 11
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*      Warm-boot, simply branch to source address     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
warmboot
       dmov   SOURCE        ; dest <-- src
       splk   #0, GREG
       lacl   DEST2
       BACC
looper splk   #0,GREG
       LACL   B2PA_3        ; load code entry into accumulator
       BACC                 ; branch to address and execute program

OLDBOOT
* COPY TO BO MEMORY, SWITCH TO MP MODE, THEN CONTINUE TO BOOT
*
       LAR    AR7,#300h     ;AR7 => B1  (300h)
       MAR    *,AR7         ;ARP => AR7
*
* MOVE THE CODE BLOCK
       RPT    #(CODE_END-CODE-1)          ; c203 bootloader is copied in B1
       BLPD   #CODE,*+                    ; BLOCK move from PM to DM
*                                         ; Code is copied in DM from 300h
       LDP    #6                          ; DP --> 300h
       LAR    AR0, #(CODE_END-CODE-1)     ; AR0 is the counter
       LAR    AR1, #300h                  ; Source address-->AR1
       MAR    *,AR1
       LACL   -#0FF00h                    ; Destination is FF00h in Prog.memory
       SACL   DEST1
COPY   LACL   *+,AR0                      ; c203 bootloader is copied in FF00h
```

```
       SACL   CODEWORD1
       LACL   DEST1
       TBLW   CODEWORD1
       ADD    #1
       SACL   DEST1
       BANZ   COPY,AR1
       SPLK   #0FF18h, 0h              ; fix to modify loop return address
       LACL   #0FF24h                  ; Write FF18h in FF24h of Prog.memory
       TBLW   300h                     ; This is required to patch the "loop"
       MAR    *,AR1                    ; address in the original c203 bootloader
       LDP    #0                       ; after relocation to FF00h
       B      0FF00h
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*      BOOT LOAD FROM 8-BIT MEMORY, MS BYTE IS FIRST  *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
* change to MP mode from MC mode
CODE
        SPLK    0007h, TEMP     ; set to microprocessor mode
        OUT     TEMP,PMST       ; write to PMST register, SARAM mapped in
                                ; program and data (SARAM is internal)
*
*  Determine destination address
*
        SPLK    #80h,GREG       ; LOCATIONS 8000-FFFFH are in global data space
        LAR     AR1,#SRC        ; AR1 points to Global address 8000h
        LACC    *+,8            ; Load ACC with high byte and shift 8 bits
        SACL    HBYTE           ; store high byte
        LACL    *+              ; load ACC with low byte of destination
        AND     #0FFH           ; Mask off upper 24 bits.
        OR      HBYTE           ; OR ACC with high byte to form 16 bit
                                ; destination address
        SACL    DEST            ; store destination address in PM
        SACL    B2PA_3          ; (71h - Program start address)
*
*  Determine length of code to be transferred
*
        LACC    *+,8            ; Load ACC with high byte and shift 8 bits
        SACL    HBYTE           ; store high byte
        LACL    *+              ; load ACC with low byte of length
        AND     #0FFH           ; Mask off upper 24 bits.
        OR      HBYTE           ; or ACC with hbyte to form 16 bit length
        SACL    LENGTH          ; store length
        LAR     AR0,LENGTH      ; load aro with length to be used for banz
*
*  Transfer code
*
LOOP    LACC    *+,8            ; Load ACC with high byte of code & shift 8 bits
        SACL    HBYTE           ; store high byte
        LACL    *+,AR0
        AND     #0FFH           ;
        OR      HBYTE           ; OR ACC with hbyte to form 16 bit code word
        SACL    CODEWORD
        LACL    DEST
```

```
        TBLW    CODEWORD
        ADD     #1
        SACL    DEST
        BANZ    LOOP,AR1        ; determine if end of code is reached
        splk    #0,GREG         ; Remove global memory
        LACL    B2PA_3          ; load code entry into ACCumulator
        BACC                    ; branch to address and execute program
CODE_END

PAR08: ;****************** 8–BIT EPROM BOOTLOADER CODE BEGINS *************
*  Determine destination address
*
        SPLK    #80h,GREG       ; LOCATIONS 8000–FFFFH are in global data space
        LAR     AR1,SOURCE      ; AR1 points to starting address of EPROM in
                                ; global memory space
TOP     LACC    *+,8            ; Load ACC with high byte and shift 8 bits
        SACL    HBYTE           ; store high byte
        LACL    *+              ; load ACC with low byte of destination
        AND     #0FFH           ; Mask off upper 24 bits.
        OR      HBYTE           ; OR ACC with high byte to form 16 bit
                                ; destination address ––> ACC
        bit     FDEST,15        ; FDEST = 1 in first pass
        bcnd    skip5,ntc
        splk    #0, FDEST       ; FDEST = 0 from second pass
        SACL    B2PA_3          ; Save final destination address to jump to.
skip5   SACL    DEST            ; Store destination address
        bit     nextsect,15     ; check to see if through at least one section
        bcnd    cont1,ntc       ; nextsect = 0 in first pass
        lacl    DEST
        and     #0FFFFh
        bcnd    looper,eq       ; if word is 0000h, booting is done
        splk    #0,nextsect
cont1
*
*  Determine length of code to be transferred
*
        LACC    *+,8            ; Load ACC with high byte and shift 8 bits
        SACL    HBYTE           ; store high byte
        LACL    *+              ; load ACC with low byte of length
        AND     #0FFH           ; Mask off upper 24 bits.
        OR      HBYTE           ; OR ACC with high byte to form 16 bit length
        SACL    LENGTH          ; store length
        LAR     AR0,LENGTH      ; load AR0 with length to be used for banz
*
*  Transfer code
*
LOOP1   LACC    *+,8            ; Load ACC with high byte of code & shift 8 bits
        SACL    HBYTE           ; store high byte
        LACL    *+,AR0
        AND     #0FFH
        OR      HBYTE           ; OR ACC with hbyte to form 16 bit code word
        SACL    CODEWORD
        LACL    DEST
        TBLW    CODEWORD
```

```
        ADD     #1
        SACL    DEST
        BANZ    LOOP1,AR1       ; determine if end of code is reached
        call B2_init            ; reinitialize for next section
        splk #1, nextsect       ; flag to check for another section
        B    TOP
*** 8-bit EPROM bootloader code ends ***
PAR16: ; ************** 16-bit EPROM BOOTLOADER CODE BEGINS ***************
*  Determine destination address
*
        SPLK    #80h,GREG       ; LOCATIONS 8000-FFFFH are in global data space
        LAR     AR1,SOURCE      ; AR1 points to starting address of EPROM in
                                ; global memory space
TOP1    LACC    *+              ; Load ACC with destination address
        bit     FDEST,15        ; FDEST = 1 in first pass
        bcnd    skip2,ntc
        splk    #0, FDEST       ; FDEST = 0 from second pass
        SACL    B2PA_3          ; save final destination address to jump to
skip2   SACL    DEST            ; store destination address
        bit     nextsect,15     ; nextsect = 0 in first pass
        bcnd    cont2,ntc
        lacl    DEST
        and     #0FFFFh
        bcnd    looper,eq
        splk    #0,nextsect
cont2
*
*  Determine length of code to be transferred
*
        LACC    *+                  ; Load ACC with length of section
        SACL    LENGTH              ; store length
        LAR     AR0,LENGTH          ; load aro with length to be used for banz
*
*  Transfer code
*
LOOP2   LACC    *+, AR0             ; Load ACC with high byte of code
        SACL    CODEWORD
        LACL    DEST
        TBLW    CODEWORD
        ADD     #1
        SACL    DEST
        BANZ    LOOP2,AR1           ; determine if end of code is reached
        call    B2_init             ; reinitialize for next section
        splk    #1, nextsect        ; flag to check for another section
        B       TOP1
*** 16-bit EPROM bootloader code ends ***
ASP: ; *********** ASYNCH. SERIAL PORT (UART) BOOTLOADER CODE BEGINS *********
* Function: 2xx Serial loader module by polling DR bit     *
*                                                          *
*           Receive data format :                          *
*           Header :                                       *
*                    start address         1st word        *
*                    Program code/length   2nd word        *
*                    Program code/data     from 3rd word   *
```

```
*           After data load the PC jumps to the            *
*           Destination/Load/Run address.                  *
* UART initialization with autobaud enable

        ldp #0
        splk #0c0a0h,B2S_0      ; reset the UART by writing 0
        out B2S_0, aspcr        ; Enable Auto baud detect & Rcv interrupt
        splk #0e0a0h,B2S_0      ; CAD=1, 1 stop bit
        out B2S_0,aspcr
        splk #4fffh,B2S_0       ; Clear ADC & BI bits
        out B2S_0,iosr          ; enable auto baud
uart:   in B2S_0,iosr
        bit B2S_0,7             ; check DR bit to see if any new character
        bcnd uart,ntc           ; is available in the ADTR
        in B2S_0,aspcr
        bit B2S_0,10            ; Check CAD =1
        bcnd nrcv,ntc           ; If 0 , start receive, autobaud done
        in B2S_1,iosr           ; load input status from iosr
        bit B2S_1,1             ; check if auto baud bit is set,else return
        bcnd nauto,ntc          ; and wait for Auto baud detect receive
        splk #4000h,B2S_1       ; Auto baud detect done
        out B2S_1,iosr          ; clear ADC
        splk #0e080h,B2S_1
        out B2S_1, aspcr        ; Disable CAD bit/ auto baud
        in B2S_1,adtr           ; Dummy read to discard ”a”
        out B2S_1,adtr          ; Echo back ”a”
nauto:  in B2S_1,adtr           ; Dummy read to clear UART rx buffer
        b skip1                 ; Exit and wait for ”a”
skip1:  splk #6600h,B2S_0
        out B2S_0,iosr          ; Clear all Interrupt sources
        B uart
nrcv:
*  Begin receiving user code
        setc CNF                ; map B0 to program space
        call B2_init            ;
pwait:
        in B2S_0,iosr           ; Load input status from iosr
        bit B2S_0,7             ; bit 8 in the data
        bcnd pwait,ntc          ; IF DR=0 no echo, return
        call pnrcv              ;
        bit B2FM_8,15           ; Wait until Data_move ready flag
        bcnd pwait,ntc
        lacl B2PA_2             ; Load destination address
        tblw B2PD_5             ; Move data to the current destination address
        add #1                  ; Increment destination address+1
        sacl B2PA_2             ; save next destination address
        banz pwait,*-
* check if next section, need to read next 16 bit word, if ”0000” then a
* section follows else program branches to address saved in B2PA_3.
        call B2_init            ; reinitialize for next section
        splk #1, nextsect       ; flag to check for another section
        B pwait
pnrcv:
        mar *,ar1               ; Valid UART data, Point to Word index reg.
```

```
        bit B2D_6,15             ; Check if bit0 of word index =1,low byte
        bcnd plbyte,tc           ; received!
        in B2S_1,adtr            ; No, Hi byte received!
        out B2S_1,adtr           ; Echo receive data
        lacc B2S_1,8             ; Align to upper byte
        sacl B2D_7               ; Save aligned word
        mar *+                   ; Increment Word Index
        sar ar1,B2D_6            ; Store high_byte flag
        splk #0,B2FM_8           ; Reset Data/word move flag as only hi-byte recd!
        b pskip                  ; wait for next byte
plbyte:
        in B2S_0,adtr            ; Receive second byte/low byte
*       out B2S_0,adtr           ; Echo received data
        lacc B2S_0,0
        and #0ffh                ; Clear high byte
        or B2D_7                 ; Add high byte to the word
        sacl B2PD_5              ; store 16-bit word at ar1
        mar *+                   ; 1+
        sar ar1,B2D_6            ; Save the count
        bit nextsect,15          ; check for next section
        bcnd cont,ntc            ; if not zero, continue, else check for 0
        lacl B2PD_5              ; load first word
        and #0FFFFh
        bcnd looper,eq           ; if 0 done, else
        splk #0,nextsect         ; reset next sect flag for next pass
cont    bit B2FH_9,15            ; Check Header_done flag
        bcnd psmove,tc           ; No, if 2 words received update Data_move flag
        lar ar0,#2
        cmpr 0
        bcnd pword2,ntc
        bit  FDEST,15            ; test to determine if this is first pass
        bcnd skip,ntc            ; skip if this is 2nd section onward
        splk #0, FDEST           ; if yes reset flag
        sacl B2PA_3              ; Store DESTINATION address to JUMP TO
skip    sacl B2PA_2              ; Save data buffer address
        b pskip                  ;
pword2:
        lar ar0,#4               ; Check if 4 words recvd, update program length
        cmpr 0                   ; Program length register
        bcnd pskip,ntc           ; Else exit
        lar ar2,B2PD_5           ; Yes received!,Load PM length in AR2
        sar ar2, B2PL_4          ; Save program length
        splk #1,B2FH_9           ; Set Header_done flag
        b pskip
psmove:
        mar *,ar2
        splk #1h,B2FM_8          ; Set UART Data_move ready flag
pskip:
        splk #0020h, ifr         ; Clear interrupt in ifr!
        ret
B2_init:
        lacc #0
        lar ar1,#B2              ; Point B2_RAM start address
        mar *,ar1
```

```
        rpt #16
        sacl *+                 ; Clear B2 memory
        lar ar1,#00h            ; Clear pointers
        lar ar2,#00h            ;
        lar ar3,#00h
        ret
*** Asynch. serial port (UART) bootloader code ends ***
*********************************************************
*  SERIAL BOOTLOAD (SSP 8/16 bit,UART), PARALLEL I/O    *
*********************************************************
ser_io
        bit     brs,b4          ; test bit 4 of configuration word
        bcnd    ASP,TC          ; If set, branch to UART bootloader
        bit     brs,b3          ; test bit 3 of configuration word
        bcnd    io,tc           ; If set, branch to Parallel I/O bootloader
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from Synchronous serial port  (SSP)      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ser
        bit     brs,b2          ; test bit 2 of configuration word
        bcnd    bit8,ntc        ; if 0, then 8-bit mode, else 16-bit mode
*********** 16-BIT SYNCH. SERIAL PORT (SSP) BOOTLOADER CODE BEGINS *********
*           After data load the PC jumps to the Destination   *
*           /Load/Run address.                                *

        setc CNF                ; Block B0 in PM
        ldp  #0h                ; set DP=0
        setc INTM               ; Disable all interrupts
        call  B2_init
        splk #0,nextsect
        splk #1,FDEST           ; FLAG to determine address of code entry
*SSP initialization
sspld:  splk #0c00ah,B2S_0      ; Initialize SSP in Burst mode, in reset
        out B2S_0,sspcr         ; External Clocks, 16 bit word
        splk #0c03ah, B2S_0     ; Interrupt on 1 word in FIFO, Internal FSX
        out B2S_0, sspcr        ; take port out of reset
wait:   in B2S_0,sspcr
        bit B2S_0,3             ; poll RFNE bit to see if data received
        bcnd wait,ntc
        call codrx
        bit B2FM_8,15           ; Wait until Data_move ready flag
        bcnd wait,ntc
        splk #0,B2FM_8
        lacl B2PA_2             ; Load destination address
        tblw B2PD_5             ; Move data to the current destination address
        add #1                  ; Increment destination address+1
        sacl B2PA_2             ; save next destination address
        banz wait,*-            ; decrement length counter
* check if next section, need to read next 16 bit word, if not ”0000” then a
* section follows else program branches to address saved in B2PA_3.
        call B2_init            ; reinitialize for next section
        splk #1, nextsect       ; flag to check for another section
        B wait
* SSP loader code!
```

```
codrx:
        in B2S_0,sdtr           ; Read received data/Load Scratch RAM
        out B2S_0,sdtr          ; Echo received data
        bit nextsect,15         ; check for next section/BIT 0 of nextsect
        bcnd contx,ntc          ; if not zero, continue, else check for 0
        lacl  B2S_0
*       lacl B2PD_5             ; load first word
        and #0FFFFh
        lar  ar7, #9999h
        bcnd looper,eq          ; if 0 done, else
        splk #0,nextsect        ; reset next sect flag for next pass
contx   mar *,ar3               ; Set Word index register as AR3
        mar *+                  ; Increment word index
        lar ar0,#1              ; If word index =1 save Program start address
        cmpr 0
        bcnd pmad,tc
        lar ar0,#2              ; If index =2 save Program length
        cmpr 0                  ; Compare if (AR3)=(AR0). TC=1, if true
        bcnd plen,tc            ; True in second pass
        lacc B2S_0,0
        sacl B2PD_5,0           ; Store received word
        splk #1h,B2FM_8         ; Set SSP Data_move ready flag
        b skip7,ar2
pmad:   lacc B2S_0,0            ; Store destination start address in ACC
        bit  FDEST,15           ; test to determine if this is first pass
        bcnd skip6,ntc          ; skip if this is 2nd section onward
        splk #0, FDEST          ; if yes reset flag
        sacl B2PA_3             ; Store DESTINATION address to JUMP TO
skip6   sacl B2PA_2             ; Save data buffer address
        b skip7,ar2             ;

plen:   lar ar2,B2S_0           ; Store Program length at B2PL_4
        sar ar2,B2PL_4
skip7:
        ret
*** 16-bit Synch. serial port (SSP) bootloader code ends ***
*********** 8-BIT SYNCH. SERIAL PORT (SSP) BOOTLOADER CODE BEGINS *********
bit8
* Function: F2xx Serial loader module                       *
*                                                           *
*           Receive data format :                           *
*           Header :                                        *
*                    start address          1st word        *
*                    Program code/length    2nd word        *
*                    Program code/data      from 3rd word   *
*           After data load the PC jumps to the             *
*           Destination/Load/Run address.                   *
        .title  " Serial loader"  ; Title
        setc CNF                ; Block B0 in PM
        ldp #0h                 ; set DP=0
        setc INTM               ; Disable all interrupts
        call B2_init
        splk #0,nextsect
        splk #1,FDEST           ; FLAG to determine address of code entry
```

```
*SSP initialization
sspld1  splk #0c00ah,B2S_0      ; Initialize SSP in Burst mode, in reset
        out B2S_0,sspcr         ; External Clocks, 16 bit word
        splk #0c03ah, B2S_0     ; Interrupt on 1 word in FIFO, external FSX
        out B2S_0, sspcr        ; take port out of reset
        splk #0001h, B2S_0
        out B2S_0,sspst         ; 8 bit mode
*       splk #8h,imr            ; Enable SSP RX interrupt only
pwait1:
        in B2S_0,sspcr          ; Load input status from sspcr
        bit B2S_0,3             ; Poll RFNE bit
        bcnd pwait1,ntc         ; IF DR=0 no echo, return
        call pnrcv1             ;
        bit B2FM_8,15           ; Wait until Data_move ready flag
        bcnd pwait1,ntc
        lacl B2PA_2             ; Load destination address
        tblw B2PD_5             ; Move data to the current destination address
        add #1                  ; Increment destination address+1
        sacl B2PA_2             ; save next destination address
        banz pwait1,*-
* check if next section, need to read next 16 bit word, if not "0000" then a
* section follows else program branches to address saved in B2PA_3.
        call B2_init            ; reinitialize for next section
        splk #1, nextsect       ; flag to check for another section
        B pwait1
pnrcv1:
        mar *,ar1               ; Valid  data, Point to Word index reg.
        bit B2D_6,15            ; Check if bit0 of word index =1,low byte
        bcnd lbyte,tc           ; received!
        in B2S_1,sdtr           ; No, Hi byte received!
        out B2S_1,sdtr          ; Echo receive data
        lacc B2S_1,8            ; Align to upper byte
        sacl B2D_7              ; Save aligned word
        mar *+                  ; Increment Word Index
        sar ar1,B2D_6           ; Store high_byte flag
        splk #0,B2FM_8          ; Reset Data/word move flag as only hi-byte recd!
        b pskip8                ; wait for next byte
lbyte:
        in B2S_0,sdtr           ; Receive second byte/low byte
*       out B2S_0,sdtr          ; Echo received data
        lacc B2S_0,0
        and #0ffh               ; Clear high byte
        or B2D_7                ; Add high byte to the word
        sacl B2PD_5             ; store 16-bit word at ar1
        mar *+                  ; 1+
        sar ar1,B2D_6           ; Save the count
        bit nextsect,15         ; check for next section
        bcnd cont9,ntc          ; if not zero, continue, else check for 0
        lacl B2PD_5             ; load first word
        and #0FFFFh
        bcnd looper,eq          ; if 0 done, else
        splk #0,nextsect        ; reset next sect flag for next pass
cont9   bit B2FH_9,15           ; Check Header_done flag
        bcnd psmove0,tc         ; No, if 2 words received update Data_move flag
```

```
        lar ar0,#2
        cmpr 0
        bcnd word2,ntc
        bit  FDEST,15           ; test to determine if this is first pass
        bcnd skipe,ntc          ; skip if this is 2nd section onward
        splk #0, FDEST          ; if yes reset flag
        sacl B2PA_3             ; Store DESTINATION address to JUMP TO
skipe   sacl B2PA_2             ; Save data buffer address
        b pskip8                ;
word2:
        lar ar0,#4              ; Check if 4 words recvd, update program length
        cmpr 0                  ; Program length register
        bcnd pskip8,ntc         ; Else exit
        lar ar2,B2PD_5          ; Yes received!,Load PM length in AR2
        sar ar2, B2PL_4         ; Save program length
        splk #1,B2FH_9          ; Set Header_done flag
        b pskip8
psmove0:
        mar *,ar2
        splk #1h,B2FM_8         ; Set UART Data_move ready flag
pskip8:
        ret
*** 8–bit Synch. serial port (SSP) bootloader code ends ***
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from parallel I/O port (port 1) –8/16 bit parallel I/O *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
io
        splk    #0,GREG         ; disable global space
        bit     brs,b2          ; test bit #2 of configuration word
        bcnd    pasync08,ntc    ; if reset, use 8–bit mode
*********** 16–BIT PARALLEL I/O BOOTLOADER CODE BEGINS ****************
pasync16
       mar     *,ar1
TOP3   call   handshake
       IN     DEST,1              ; read word from port 1 to destination
       LACL   DEST
       bit    FDEST,15
       bcnd   skip3,ntc
       splk   #0, FDEST
       SACL   B2PA_3              ; save final destination address to jump to
skip3  SACL   DEST                ; store destination address
       bit    nextsect,15
       bcnd   cont3,ntc
       lacl   DEST
       and    #0FFFFh
       bcnd   looper,eq
       splk   #0,nextsect
cont3
       call   handshake
       IN     LENGTH,1            ; read word from port 1 to length
       lar    ar1,LENGTH          ; ar1 <–– code length
       lacl   DEST                ; ACC <–– destination address
loop16 call   handshake
       IN     TEMP,1              ; read word from port 1 to temp
```

```
       setc   xf                    ; acknowledge word as soon as it's read
       nop                          ; delay between xf and write
       nop
       tblw   TEMP                  ; write word to destination
       add    #1                    ; increment destination address
       banz   loop16,*-             ; loop if ar1 is not zero
       call   B2_init               ; reinitialize for next section
       splk   #1, nextsect          ; flag to check for another section
       B      TOP3
*** 16-bit Parallel I/O bootloader code ends ***
******** 8-BIT PARALLEL I/O BOOTLOADER CODE BEGINS -  MS byte first *******
pasync08
       mar    *,ar1
TOP4   call   handshake
       IN     TEMP,1                ; read I/O port 1
       lacc   TEMP,8                ; read high byte from port
       sacl   DEST
       call   handshake
       IN     TEMP,1
       lacl   TEMP                  ; read low byte from port
       and    #0ffh                 ; clear upper byte
       or     DEST                  ; combine high and low byte

       bit    FDEST,15
       bcnd   skip4,ntc
       splk   #0, FDEST
       SACL   B2PA_3                ; save final destination address to jump to
skip4  SACL   DEST                  ; store destination address
       bit    nextsect,15
       bcnd   cont4,ntc
       lacl   DEST
       and    #0FFFFh
       bcnd   looper,eq
       splk   #0,nextsect
cont4
       call   handshake
       IN     TEMP,1
       lacc   TEMP,8                ; read high byte from port
       sacl   LENGTH                ; save high byte
       call   handshake
       IN     TEMP,1
       lacl   TEMP                  ; read low byte from port
       and    #0ffh                 ; clear upper byte
       or     LENGTH                ; combine high and low byte
       sacl   LENGTH                ; save code length
       LAR    ar1,LENGTH            ; ar1 <-- code length
       lacl   DEST
       sacl   DEST2                 ; DEST2 <-- destination address
loop08 call   handshake
       IN     TEMP,1
       lacc   TEMP,8                ; read high byte from port
       sacl   TEMP1                 ; save high byte
       call   handshake
       IN     TEMP,1
```

```
        lacl   TEMP                ; read low byte from port
        setc   xf                  ; acknowledge byte as soon as it’s read
        and    #0ffh               ; clear upper byte
        or     TEMP1               ; combine high and low byte
        sacl   TEMP1               ; save code word
        lacl   DEST2               ; DEST2 <–– destination address
        tblw   TEMP1               ; write code word to program memory
        add    #1                  ; increment destination address
        sacl   DEST2               ; save new destination address
        banz   loop08,*–           ; loop if ar1 not zero
        call   B2_init             ; reinitialize for next section
        splk   #1, nextsect        ; flag to check for another section
        B      TOP4

*** 8–bit Parallel I/O bootloader code ends ***
*       Handshake with BIO signal using XF
handshake
        setc   xf                  ; acknowledge previous data word
biohigh
        bcnd   biohigh,bio         ; wait till host sends request
        clrc   xf                  ; indicate ready to receive new data
biolow
        retc   bio                 ; wait till new data ready
        b      biolow

        .sect  ”alaw”
;***************************************************************************
;
;       CCITT expansion table
;       The table is A–law expansion table for ADI–coded samples. Please read
;       columnar values top to bottom and from left column to next right column.
;***************************************************************************
        .DEF   AEXPTAB
```

```
AEXPTAB .WORD   -688    .WORD   -1248   .WORD   -204
        .WORD   -656    .WORD   -1184   .WORD   -196
        .WORD   -752    .WORD   -1888   .WORD   -220
        .WORD   -720    .WORD   -1824   .WORD   -212
        .WORD   -560    .WORD   -2016   .WORD   -86
        .WORD   -528    .WORD   -1952   .WORD   -82
        .WORD   -624    .WORD   -1632   .WORD   -94
        .WORD   -592    .WORD   -1568   .WORD   -90
        .WORD   -944    .WORD   -1760   .WORD   -70
        .WORD   -912    .WORD   -1696   .WORD   -66
        .WORD   -1008   .WORD   -43     .WORD   -78
        .WORD   -976    .WORD   -41     .WORD   -74
        .WORD   -816    .WORD   -47     .WORD   -118
        .WORD   -784    .WORD   -45     .WORD   -114
        .WORD   -880    .WORD   -35     .WORD   -126
        .WORD   -848    .WORD   -33     .WORD   -122
        .WORD   -344    .WORD   -39     .WORD   -102
        .WORD   -328    .WORD   -37     .WORD   -98
        .WORD   -376    .WORD   -59     .WORD   -110
        .WORD   -360    .WORD   -57     .WORD   -106
        .WORD   -280    .WORD   -63     .WORD   688
        .WORD   -264    .WORD   -61     .WORD   656
        .WORD   -312    .WORD   -51     .WORD   752
        .WORD   -296    .WORD   -49     .WORD   720
        .WORD   -472    .WORD   -55     .WORD   560
        .WORD   -456    .WORD   -53     .WORD   528
        .WORD   -504    .WORD   -11     .WORD   624
        .WORD   -488    .WORD   -9      .WORD   592
        .WORD   -408    .WORD   -15     .WORD   944
        .WORD   -392    .WORD   -13     .WORD   912
        .WORD   -440    .WORD   -3      .WORD   1008
        .WORD   -424    .WORD   -1      .WORD   976
        .WORD   -2752   .WORD   -7      .WORD   816
        .WORD   -2624   .WORD   -5      .WORD   784
        .WORD   -3008   .WORD   -27     .WORD   880
        .WORD   -2880   .WORD   -25     .WORD   848
        .WORD   -2240   .WORD   -31     .WORD   344
        .WORD   -2112   .WORD   -29     .WORD   328
        .WORD   -2496   .WORD   -19     .WORD   376
        .WORD   -2368   .WORD   -17     .WORD   360
        .WORD   -3776   .WORD   -23     .WORD   280
        .WORD   -3648   .WORD   -21     .WORD   264
        .WORD   -4032   .WORD   -172    .WORD   312
        .WORD   -3904   .WORD   -164    .WORD   296
        .WORD   -3264   .WORD   -188    .WORD   472
        .WORD   -3136   .WORD   -180    .WORD   456
        .WORD   -3520   .WORD   -140    .WORD   504
        .WORD   -3392   .WORD   -132    .WORD   488
        .WORD   -1376   .WORD   -156    .WORD   408
        .WORD   -1312   .WORD   -148    .WORD   392
        .WORD   -1504   .WORD   -236    .WORD   440
        .WORD   -1440   .WORD   -228    .WORD   424
        .WORD   -1120   .WORD   -252    .WORD   2752
        .WORD   -1056   .WORD   -244    .WORD   2624
```

```
.WORD    3008
.WORD    2880
.WORD    2240
.WORD    2112
.WORD    2496
.WORD    2368
.WORD    3776
.WORD    3648
.WORD    4032
.WORD    3904
.WORD    3264
.WORD    3136
.WORD    3520
.WORD    3392
.WORD    1376
.WORD    1312
.WORD    1504
.WORD    1440
.WORD    1120
.WORD    1056
.WORD    1248
.WORD    1184
.WORD    1888
.WORD    1824
.WORD    2016
.WORD    1952
.WORD    1632
.WORD    1568
.WORD    1760
.WORD    1696
.WORD    43
.WORD    41
.WORD    47
.WORD    45
.WORD    35
.WORD    33
.WORD    39
.WORD    37
.WORD    59
.WORD    57
.WORD    63
.WORD    61
.WORD    51
.WORD    49
.WORD    55
.WORD    53
.WORD    11
.WORD    9
.WORD    15
.WORD    13
.WORD    3
.WORD    1
.WORD    7
.WORD    5
        .WORD    27
        .WORD    25
        .WORD    31
        .WORD    29
        .WORD    19
        .WORD    17
        .WORD    23
        .WORD    21
        .WORD    172
        .WORD    164
        .WORD    188
        .WORD    180
        .WORD    140
        .WORD    132
        .WORD    156
        .WORD    148
        .WORD    236
        .WORD    228
        .WORD    252
        .WORD    244
        .WORD    204
        .WORD    196
        .WORD    220
        .WORD    212
        .WORD    86
        .WORD    82
        .WORD    94
        .WORD    90
        .WORD    70
        .WORD    66
        .WORD    78
        .WORD    74
        .WORD    118
        .WORD    114
        .WORD    126
        .WORD    122
        .WORD    102
        .WORD    98
        .WORD    110
        .WORD    106
        .sect    ”ulaw”
;************************
;
;  CCITT mu–law Expansion
  Table
;
;************************
        .DEF   UEXPTAB
UEXPTAB .WORD  0e0a1h
        .WORD  0e1a1h
        .WORD  0e2a1h
        .WORD  0e3a1h
        .WORD  0e4a1h
        .WORD  0e5a1h
.WORD  0e6a1h
.WORD  0e7a1h
.WORD  0e8a1h
.WORD  0e9a1h
.WORD  0eaa1h
.WORD  0eba1h
.WORD  0eca1h
.WORD  0eda1h
.WORD  0eea1h
.WORD  0efa1h
.WORD  0f061h
.WORD  0f0e1h
.WORD  0f161h
.WORD  0f1e1h
.WORD  0f261h
.WORD  0f2e1h
.WORD  0f361h
.WORD  0f3e1h
.WORD  0f461h
.WORD  0f4e1h
.WORD  0f561h
.WORD  0f5e1h
.WORD  0f661h
.WORD  0f6e1h
.WORD  0f761h
.WORD  0f7e1h
.WORD  0f841h
.WORD  0f881h
.WORD  0f8c1h
.WORD  0f901h
.WORD  0f941h
.WORD  0f981h
.WORD  0f9c1h
.WORD  0fa01h
.WORD  0fa41h
.WORD  0fa81h
.WORD  0fac1h
.WORD  0fb01h
.WORD  0fb41h
.WORD  0fb81h
.WORD  0fbc1h
.WORD  0fc01h
.WORD  0fc31h
.WORD  0fc51h
.WORD  0fc71h
.WORD  0fc91h
.WORD  0fcb1h
.WORD  0fcd1h
.WORD  0fcf1h
.WORD  0fd11h
.WORD  0fd31h
.WORD  0fd51h
.WORD  0fd71h
.WORD  0fd91h
```

```
.WORD 0fdb1h
.WORD 0fdd1h
.WORD 0fdf1h
.WORD 0fe11h
.WORD 0fe29h
.WORD 0fe39h
.WORD 0fe49h
.WORD 0fe59h
.WORD 0fe69h
.WORD 0fe79h
.WORD 0fe89h
.WORD 0fe99h
.WORD 0fea9h
.WORD 0feb9h
.WORD 0fec9h
.WORD 0fed9h
.WORD 0fee9h
.WORD 0fef9h
.WORD 0ff09h
.WORD 0ff19h
.WORD 0ff25h
.WORD 0ff2dh
.WORD 0ff35h
.WORD 0ff3dh
.WORD 0ff45h
.WORD 0ff4dh
.WORD 0ff55h
.WORD 0ff5dh
.WORD 0ff65h
.WORD 0ff6dh
.WORD 0ff75h
.WORD 0ff7dh
.WORD 0ff85h
.WORD 0ff8dh
.WORD 0ff95h
.WORD 0ff9dh
.WORD 0ffa3h
.WORD 0ffa7h
.WORD 0ffabh
.WORD 0ffafh
.WORD 0ffb3h
.WORD 0ffb7h
.WORD 0ffbbh
.WORD 0ffbfh
.WORD 0ffc3h
.WORD 0ffc7h
.WORD 0ffcbh
.WORD 0ffcfh
.WORD 0ffd3h
.WORD 0ffd7h
.WORD 0ffdbh
.WORD 0ffdfh
.WORD 0ffe2h
.WORD 0ffe4h
.WORD 0ffe6h
.WORD 0ffe8h
.WORD 0ffeah
.WORD 0ffech
.WORD 0ffeeh
.WORD 0fff0h
.WORD 0fff2h
.WORD 0fff4h
.WORD 0fff6h
.WORD 0fff8h
.WORD 0fffah
.WORD 0fffch
.WORD 0fffeh
.WORD 00000h
.WORD 01f5fh
.WORD 01e5fh
.WORD 01d5fh
.WORD 01c5fh
.WORD 01b5fh
.WORD 01a5fh
.WORD 0195fh
.WORD 0185fh
.WORD 0175fh
.WORD 0165fh
.WORD 0155fh
.WORD 0145fh
.WORD 0135fh
.WORD 0125fh
.WORD 0115fh
.WORD 0105fh
.WORD 00f9fh
.WORD 00f1fh
.WORD 00e9fh
.WORD 00e1fh
.WORD 00d9fh
.WORD 00d1fh
.WORD 00c9fh
.WORD 00c1fh
.WORD 00b9fh
.WORD 00b1fh
.WORD 00a9fh
.WORD 00a1fh
.WORD 0099fh
.WORD 0091fh
.WORD 0089fh
.WORD 0081fh
.WORD 007bfh
.WORD 0077fh
.WORD 0073fh
.WORD 006ffh
.WORD 006bfh
.WORD 0067fh
.WORD 0063fh
.WORD 005ffh
.WORD 005bfh
.WORD 0057fh
.WORD 0053fh
.WORD 004ffh
.WORD 004bfh
.WORD 0047fh
.WORD 0043fh
.WORD 003ffh
.WORD 003cfh
.WORD 003afh
.WORD 0038fh
.WORD 0036fh
.WORD 0034fh
.WORD 0032fh
.WORD 0030fh
.WORD 002efh
.WORD 002cfh
.WORD 002afh
.WORD 0028fh
.WORD 0026fh
.WORD 0024fh
.WORD 0022fh
.WORD 0020fh
.WORD 001efh
.WORD 001d7h
.WORD 001c7h
.WORD 001b7h
.WORD 001a7h
.WORD 00197h
.WORD 00187h
.WORD 00177h
.WORD 00167h
.WORD 00157h
.WORD 00147h
.WORD 00137h
.WORD 00127h
.WORD 00117h
.WORD 00107h
.WORD 000f7h
.WORD 000e7h
.WORD 000dbh
.WORD 000d3h
.WORD 000cbh
.WORD 000c3h
.WORD 000bbh
.WORD 000b3h
.WORD 000abh
.WORD 000a3h
.WORD 0009bh
.WORD 00093h
.WORD 0008bh
.WORD 00083h
.WORD 0007bh
.WORD 00073h
```

```
.WORD  0006bh
.WORD  00063h
.WORD  0005dh
.WORD  00059h
.WORD  00055h
.WORD  00051h
.WORD  0004dh
.WORD  00049h
.WORD  00045h
.WORD  00041h
.WORD  0003dh
.WORD  00039h
.WORD  00035h
.WORD  00031h
.WORD  0002dh
.WORD  00029h
.WORD  00025h
.WORD  00021h
.WORD  0001eh
.WORD  0001ch
.WORD  0001ah
.WORD  00018h
.WORD  00016h
.WORD  00014h
.WORD  00012h
.WORD  00010h
.WORD  0000eh
.WORD  0000ch
.WORD  0000ah
.WORD  00008h
.WORD  00006h
.WORD  00004h
.WORD  00002h
.WORD  00000h
```

```
Common header file:
Filename: sldrv201.h
        .mmregs
; Memory variables specific to flash algorithms
****************************************************
BASE       .set    068h           ; Base address for variables
B2_0       .set    BASE+0         ; can be changed to relocate
B2_1       .set    BASE+1         ; variable space in RAM
B2_2       .set    BASE+2
B2_3       .set    BASE+3
B2_4       .set    BASE+4
B2_5       .set    BASE+5
B2_6       .set    BASE+6
nextsect   .set    BASE+7
FDEST      .set    BASE+8
B2PA_3     .set    BASE+9         ; Program start address

* Variables for Uart_loader
****************************************************
B2         .set 72h
B2S_0      .set B2+0h              ; Scratch registers
B2S_1      .set B2+1h
B2PA_2     .set B2+2h              ; Program start address
*
B2PL_4     .set B2+4h              ; Program Length
B2PD_5     .set B2+5h              ; Program Code/Data
B2D_6      .set B2+6h              ; Variables
B2D_7      .set B2+7h
B2FM_8     .set B2+8h              ; Flag for start Data move – Data_move
B2FH_9     .set B2+9h              ; Flag for Header receive – Header_done
B2FD_a     .set B2+0ah             ; Flag for data move complete – Data_ready
B2FSH      .set B2+0bh             ; High word check sum
B2FSL      .set B2+0ch             ; Low word check sum

* On-chip I/O registers

PMST       .set    0FFE4h          ;Defines SARAM in PM/DM and MP/MC bit
* SYNC PORT
sdtr        set    0fff0h
sspcr      .set    0fff1h
sspst      .set    0fff2h
* UART
adtr       .set    0fff4h
aspcr      .set    0fff5h
iosr       .set    0fff6h
brd        .set    0fff7h
```

Chapter 5

Program Control

This chapter discusses the processes and features involved in controlling the flow of a program on the ’C20x.

Program control involves controlling the order in which one or more blocks of instructions are executed. Normally, the flow of a program is sequential: the ’C20x executes instructions at consecutive program-memory addresses. At times, a program must branch to a nonsequential address and then execute instructions sequentially at that new location. For this purpose, the ’C20x supports branches, calls, returns, repeats, and interrupts.

The ’C20x also provides a power-down mode, which halts internal program flow and temporarily lowers the power requirements of the ’C20x.

Topic **Page**

5.1 Program-Address Generation 5-2

5.2 Pipeline Operation 5-7

5.3 Branches, Calls, and Returns 5-8

5.4 Conditional Branches, Calls, and Returns 5-10

5.5 Repeating a Single Instruction 5-14

5.6 Interrupts 5-15

5.7 Reset Operation 5-35

5.8 Power-Down Mode 5-40

5.1 Program-Address Generation

Program flow requires the processor to generate the next program address (sequential or nonsequential) while executing the current instruction. Program-address generation is illustrated in Figure 5–1 and summarized in Table 5–1.

Figure 5–1. Program-Address Generation Block Diagram

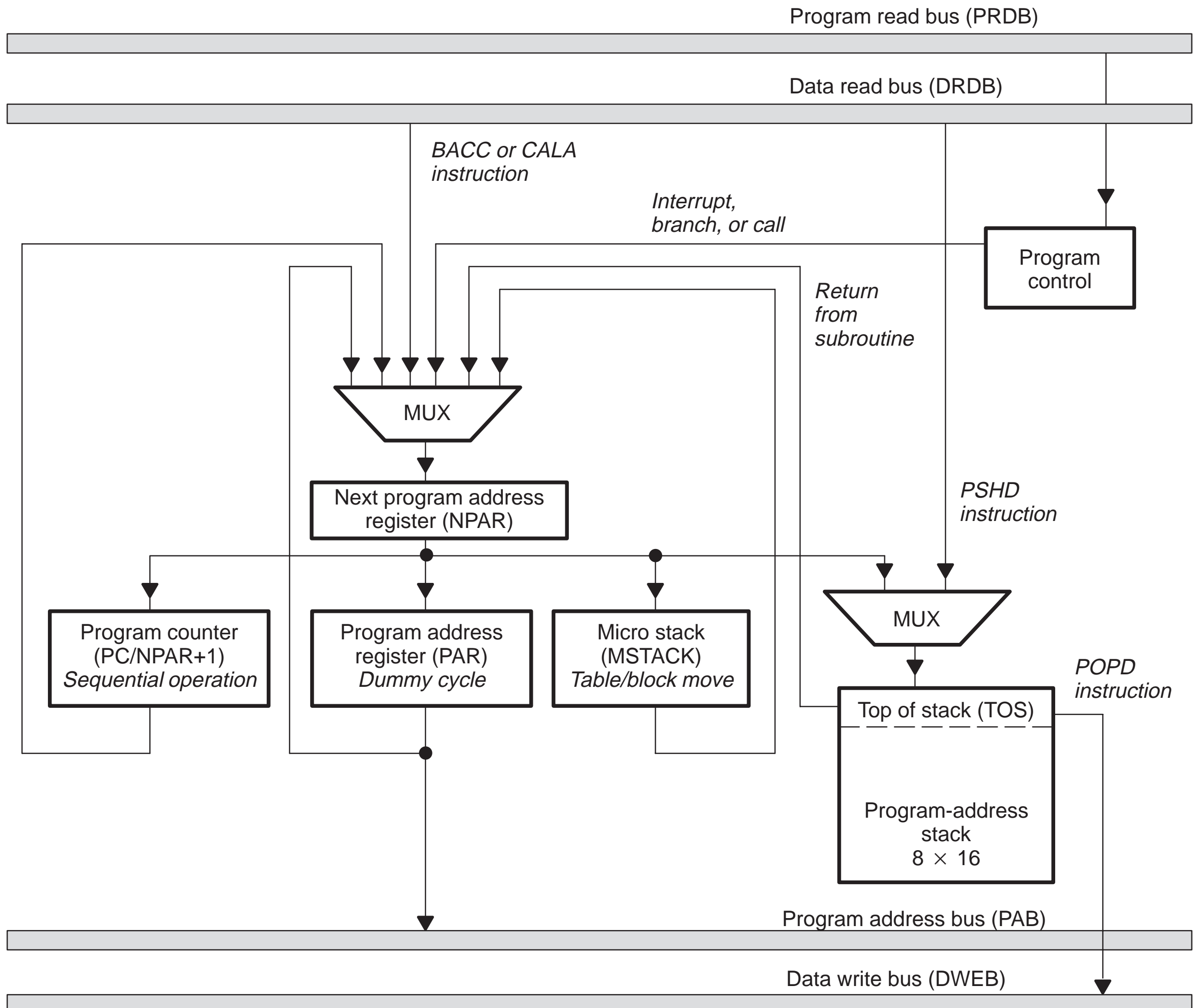


Table 5–1. Program-Address Generation Summary

Operation	Program-Address Source
Sequential operation	PC (contains program address +1)
Dummy cycle	PAR (contains program address)
Return from subroutine	Top of the stack (TOS)
Return from table move or block move	Micro stack (MSTACK)
Branch or call to address specified in instruction	Branch or call instruction by way of the program read bus (PRDB)
Branch or call to address specified in lower half of the accumulator	Low accumulator by way of the data read bus (DRDB)
Branch to interrupt service routine	Interrupt vector location by way of the program read bus (PRDB)

The ’C20x program-address generation logic uses the following hardware:

- Program counter (PC). The ’C20x has a 16-bit program counter (PC) that addresses internal and external program memory when fetching instructions.
- Program address register (PAR). The PAR drives the program address bus (PAB). The PAB is a 16-bit bus that provides program addresses for both reads and writes.
- Stack. The program-address generation logic includes a 16-bit-wide, 8-level hardware stack for storing up to eight return addresses. In addition, you can use the stack for temporary storage.
- Micro stack (MSTACK). Occasionally, the program-address generation logic uses the 16-bit-wide, 1-level MSTACK to store one return address.
- Repeat counter (RPTC). The 16-bit RPTC is used with the repeat (RPT) instruction to determine how many times the instruction following RPT is repeated.

5.1.1 Program Counter (PC)

The program-address generation logic uses the 16-bit program counter (PC) to address internal and external program memory. The PC holds the address of the next instruction to be executed. Through the program address bus (PAB), an instruction is fetched from that address in program memory and loaded into the instruction register. When the instruction register is loaded, the PC holds the next address.

The 'C20x can load the PC in a number of ways, to accommodate sequential and nonsequential program flow. Table 5–2 shows what is loaded to the PC according to the code operation performed.

Table 5–2. Address Loading to the Program Counter

Code Operation	Address Loaded to the PC
Sequential execution	The PC is loaded with PC + 1 if the current instruction has one word or PC + 2 if the current instruction has two words.
Branch	The PC is loaded with the long immediate value directly following the branch instruction.
Subroutine call and return	For a call, the address of the next instruction is pushed from the PC onto the stack, and then the PC is loaded with the long immediate value directly following the call instruction. A return instruction pops the return address back into the PC to return to the calling sequence of code.
Software or hardware interrupt	The PC is loaded with the address of the appropriate interrupt vector location. At this location is a branch instruction that loads the PC with the address of the corresponding interrupt service routine.
Computed GOTO	The content of the lower 16 bits of the accumulator is loaded into the PC. Computed GOTO operations can be performed using the BACC (branch to address in accumulator) or CALA (call subroutine at location specified by the accumulator) instructions.

5.1.2 Stack

The 'C20x has a 16-bit-wide, 8-level-deep hardware stack. The program-address generation logic uses the stack for storing return addresses when a subroutine call or interrupt occurs. When an instruction forces the CPU into a subroutine or an interrupt forces the CPU into an interrupt service routine, the return address is loaded to the top of the stack automatically; this event does not require additional cycles. When the subroutine or interrupt service routine is complete, a return instruction transfers the return address from the top of the stack to the program counter.

When the eight levels are not used for return addresses, the stack may be used for saving context data during a subroutine or interrupt service routine, or for other storage purposes.

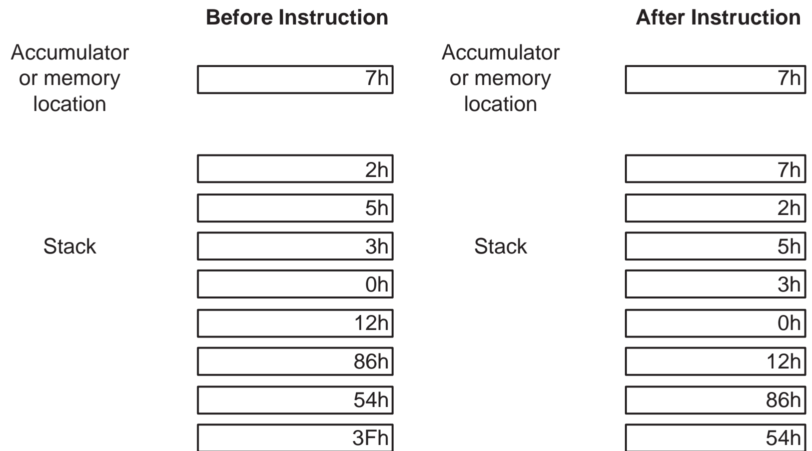
You can access the stack with two sets of instructions:

- **PUSH and POP.** The PUSH instruction copies the lower half of the accumulator to the top of the stack. The POP instruction copies the value on the top of the stack to the lower half of the accumulator.

- **PSHD and POPD.** These instructions allow you to build a stack in data memory for the nesting of subroutines or interrupts beyond eight levels. The PSHD instruction pushes a data-memory value onto the top of the stack. The POPD instruction pops a value from the top of the stack to data memory.

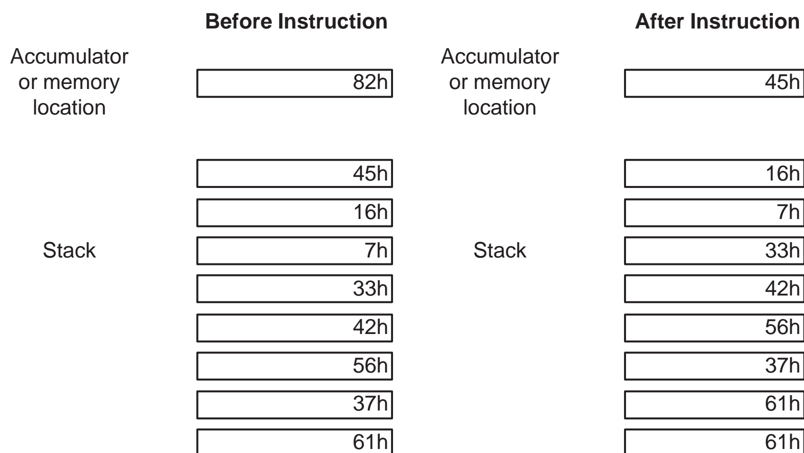
Whenever a value is pushed onto the top of the stack (by an instruction or by the address-generation logic), the content of each level is pushed down one level, and the bottom (eighth) location of the stack is lost. Therefore, data is lost (stack overflow occurs) if more than eight successive pushes occur before a pop. Figure 5–2 shows a push operation.

Figure 5–2. A Push Operation



Pop operations are the reverse of push operations. A pop operation copies the value at each level to the next higher level. Any pop after seven sequential pops yields the value that was originally at the bottom of the stack because, by then, the bottom value has been copied upward to all of the stack levels. Figure 5–3 shows a pop operation.

Figure 5–3. A Pop Operation



5.1.3 Micro Stack (MSTACK)

The program-address generation logic uses the 16-bit-wide, 1-level-deep MSTACK to store a return address before executing certain instructions. These instructions use the program-address generation logic to provide a second address in a two-operand instruction. These instructions are: BLDD, BLPD, MAC, MACD, TBLR, and TBLW. When repeated, these instructions use the PC to increment the first operand address and can use the auxiliary register arithmetic unit (ARAU) to generate the second operand address. When these instructions are used, the return address (the address of the next instruction to be fetched) is pushed onto the MSTACK. Upon completion of the repeated instruction, the MSTACK value is popped back into the program-address generation logic. The MSTACK operations are not visible to you. Unlike the stack, the MSTACK can be used only by the program-address generation logic; there are no instructions that allow you to use the MSTACK for storage.

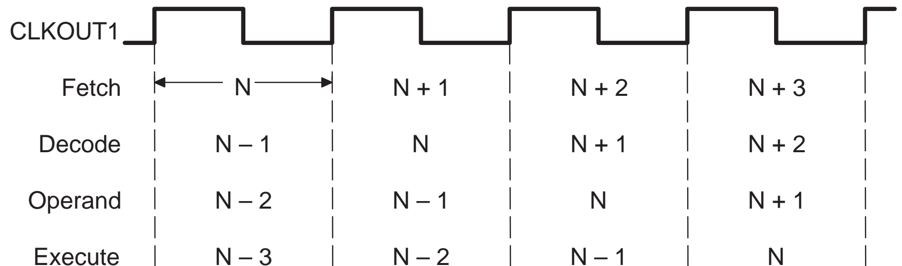
5.2 Pipeline Operation

Instruction pipelining consists of a sequence of bus operations that occur during the execution of an instruction. The ’C20x pipeline has four independent stages: instruction-fetch, instruction-decode, operand-fetch, and instruction-execute. Because the four stages are independent, these operations can overlap. During any given cycle, one to four different instructions can be active, each at a different stage of completion. Figure 5–4 shows the operation of the 4-level-deep pipeline for single-word, single-cycle instructions executing with no wait states.

The pipeline is essentially invisible to you except in the following cases:

- A single-word, single-cycle instruction immediately following a modification of the global-memory allocation register (GREG) uses the previous global map. You can prevent this by adding a NOP instruction after the instruction that writes to the GREG.
- The NORM instruction modifies the auxiliary register pointer (ARP) and uses the current auxiliary register (the one pointed to by the ARP) during the execute phase of the pipeline. If the next two instruction words change the values in the current auxiliary register or the ARP, they will do so during the instruction decode phase of the pipeline (before the execution of NORM). This would cause NORM to use the wrong auxiliary register value and the following instructions to use the wrong ARP value.

Figure 5–4. 4-Level Pipeline Operation



The CPU is implemented using 2-phase static logic. The 2-phase operation of the ’C20x CPU consists of a master phase in which all commutation logic is executed, and a slave phase in which results are latched. Therefore, sequential operations require sequential master cycles. Although sequential operations require a deeper pipeline, 2-phase operation provides more time for the computational logic to execute. This allows the ’C20x to run at faster clock rates despite having a deeper pipeline that imposes a penalty on branches and subroutine calls.

5.3 Branches, Calls, and Returns

Branches, calls, and returns break the sequential flow of instructions by transferring control to another location in program memory. A *branch* only transfers control to the new location. A *call* also saves the return address (the address of the instruction following the call) to the top of the hardware stack. Every called subroutine or interrupt service routine is concluded with a *return* instruction, which pops the return address off the stack and back into the program counter (PC).

The 'C20x has two types of branches, calls, and returns:

- Unconditional. An unconditional branch, call, or return is always executed. The unconditional branch, call, and return instructions are described in sections 5.3.1, 5.3.2, and 5.3.3, respectively.
- Conditional. A conditional branch, call, or return is executed only if certain specified conditions are met. The conditional branch, call, and return instructions are described in detail in section 5.4, *Conditional Branches, Calls, and Returns*, on page 5-10.

5.3.1 Unconditional Branches

When an unconditional branch is encountered, it is always executed. During the execution, the PC is loaded with the specified program-memory address and program execution begins at that address. The address loaded into the PC may come from either the second word of the branch instruction or the lower 16 bits of the accumulator.

By the time the branch instruction reaches the execute phase of the pipeline, the next two instruction words have already been fetched. These two instruction words are flushed from the pipeline so that they are not executed, and then execution continues at the branched-to address. The unconditional branch instructions are B (branch) and BACC (branch to location specified by accumulator).

5.3.2 Unconditional Calls

When an unconditional call is encountered, it is always executed. When the call is executed, the PC is loaded with the specified program-memory address and program execution begins at that address. The address loaded into the PC may come from either the second word of the call instruction or the lower 16 bits of the accumulator. Before the PC is loaded, the return address is saved in the stack. After the subroutine or function is executed, a return instruction loads the PC with the return address from the stack, and execution resumes at the instruction following the call.

By the time the unconditional call instruction reaches the execute phase of the pipeline, the next two instruction words have already been fetched. These two instruction words are flushed from the pipeline so that they are not executed, the return address is stored to the stack, and then execution continues at the beginning of the called function. The unconditional call instructions are CALL and CALA (call subroutine at location specified by accumulator).

5.3.3 Unconditional Returns

When an unconditional return (RET) instruction is encountered, it is always executed. When the return is executed, the PC is loaded with the value at the top of the stack, and execution resumes at that address.

By the time the unconditional return instruction reaches the execute phase of the pipeline, the next two instruction words have already been fetched. The two instruction words are flushed from the pipeline so that they are not executed, the return address is taken from the stack, and then execution continues in the calling function.

5.4 Conditional Branches, Calls, and Returns

The ’C20x provides branch, call, and return instructions that will execute only if one or more conditions are met. You specify the conditions as operands of the conditional instruction. Table 5–3 lists the conditions that you can use with these instructions and their corresponding operand symbols.

Table 5–3. Conditions for Conditional Branches, Calls, and Returns

Operand Symbol	Condition	Description
EQ	ACC = 0	Accumulator equal to zero
NEQ	ACC ≠ 0	Accumulator not equal to zero
LT	ACC < 0	Accumulator less than zero
LEQ	ACC ≤ 0	Accumulator less than or equal to zero
GT	ACC > 0	Accumulator greater than zero
GEQ	ACC ≥ 0	Accumulator greater than or equal to zero
C	C = 1	Carry bit set to 1
NC	C = 0	Carry bit cleared to 0
OV	OV = 1	Accumulator overflow detected
NOV	OV = 0	No accumulator overflow detected
BIO	$\overline{\text{BIO}}$ low	$\overline{\text{BIO}}$ pin is low
TC	TC = 1	Test/control flag set to 1
NTC	TC = 0	Test/control flag cleared to 0

5.4.1 Using Multiple Conditions

Multiple conditions can be listed as operands of the conditional instructions. If multiple conditions are listed, all conditions must be met for the instruction to execute. Note that only certain combinations of conditions are meaningful. See Table 5–4. For each combination, the conditions must be selected from Group 1 and Group 2 as follows:

- Group 1. You can select up to two conditions. Each of these conditions must be from a different category (A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time, but you cannot test GT and NEQ at the same time.

- ❑ Group 2. You can select up to three conditions. Each of these conditions must be from a different category (A or B); you cannot have two conditions from the same category. For example, you can test TC and C at the same time, but you cannot test C and NC at the same time.

Table 5–4. Groupings of Conditions

Group 1		Group 2		
Category A	**Category B**	**Category A**	**Category B**	**Category C**
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	
LT				
LEQ				
GT				
GEQ				

5.4.2 Stabilization of Conditions

A conditional instruction must be able to test the most recent values of the status bits. Therefore, the conditions cannot be considered stable until the fourth, or execution stage of the pipeline, one cycle after the previous instruction has been executed. The pipeline controller stops the decoding of any instructions following the conditional instruction until the conditions are stable.

5.4.3 Conditional Branches

A branch instruction transfers program control to any location in program memory. Conditional branch instructions are executed only when one or more user-specified conditions are met (see Table 5–3 on page 5-10). If all the conditions are met, the PC is loaded with the second word of the branch instruction, which contains the address to branch to, and execution continues at this address.

By the time the conditions have been tested, the two instruction words following the conditional branch instruction have already been fetched in the pipeline. If all the conditions are met, these two instruction words are flushed from the pipeline so that they are not executed, and then execution continues at the branched-to address. If the conditions are *not* met, the two instruction words are executed instead of the branch. Because conditional branches use

conditions determined by the execution of the previous instructions, a conditional branch takes one more cycle than an unconditional one.

The conditional branch instructions are BCND (branch conditionally) and BANZ (branch if currently selected auxiliary register is not equal to 0). The BANZ instruction is useful for implementing loops.

5.4.4 Conditional Calls

The conditional call (CC) instruction is executed only when the specified condition or conditions are met (see Table 5–3 on page 5-10). This allows your program to choose among multiple subroutines based on the data being processed. If all the conditions are met, the PC is loaded with the second word of the call instruction, which contains the starting address of the subroutine. Before branching to the subroutine, the processor stores the address of the instruction following the call instruction—the return address—to the stack. The function must end with a return instruction, which will take the return address off the stack and force the processor to resume execution of the calling program.

By the time the conditions of the conditional call instruction have been tested, the two instruction words following the call instruction have already been fetched in the pipeline. If all the conditions are met, these two instruction words are flushed from the pipeline so that they are not executed, and then execution continues at the beginning of the called function. If the conditions are *not* met, the two instructions are executed instead of the call. Because there is a wait cycle for conditions to become stable, the conditional call takes one more cycle than the unconditional one.

5.4.5 Conditional Returns

Returns are used in conjunction with calls and interrupts. A call or interrupt stores a return address to the stack and then transfers program control to a new location in program memory. The called subroutine or the interrupt service routine concludes with a return instruction, which pops the return address off the top of the stack and into the program counter (PC).

The conditional return instruction (RETC) is executed only when one or more conditions are met (see Table 5–3 on page 5-10). By using the RETC instruction, you can give a subroutine or interrupt service routine more than one possible return path. The path chosen then depends on the data being processed. In addition, you can use a conditional return to avoid conditionally branching to/around the return instruction at the end of the subroutine or interrupt service routine.

If all the conditions are met for execution of the RETC instruction, the processor loads the return address from the stack to the PC and resumes execution of the calling or interrupted program.

RETC, like RET, is a single-word instruction. However, because of the potential PC discontinuity, it operates with the same effective execution time as the conditional branch (BCND) and the conditional call (CC). By the time the conditions of the conditional return instruction have been tested, the two instruction words following the return instruction have already been fetched in the pipeline. If all the conditions are met, these two instruction words are flushed from the pipeline so that they are not executed, and then execution of the calling program continues. If the conditions are *not* met, the two instructions are executed instead of the return. Because there is a wait cycle for conditions to become stable, the conditional return takes one more cycle than the unconditional one.

5.5 Repeating a Single Instruction

The ’C20x repeat (RPT) instruction allows the execution of a single instruction N + 1 times, where N is specified as an operand of the RPT instruction. When RPT is executed, the repeat counter (RPTC) is loaded with N. RPTC is then decremented every time the repeated instruction is executed, until RPTC equals zero. RPTC can be used as a 16-bit counter when the count value is read from a data-memory location; if the count value is specified as a constant operand, it is in an 8-bit counter.

The repeat feature is useful with instructions such as NORM (normalize contents of accumulator), MACD (multiply and accumulate with data move), and SUBC (conditional subtract). When instructions are repeated, the address and data buses for program memory are free to fetch a second operand in parallel with the address and data buses for data memory. This allows instructions such as MACD and BLPD to effectively execute in a single cycle when repeated.

5.6 Interrupts

Interrupts are hardware- or software-driven signals that cause the ’C20x to suspend its current program sequence and execute a subroutine. Typically, interrupts are generated by hardware devices that need to give data to or take data from the ’C20x (for example, A/D and D/A converters and other processors). Interrupts can also signal that a particular event has taken place (for example, a timer has finished counting).

The ’C20x supports both software and hardware interrupts:

- A *software interrupt* is requested by an instruction (INTR, NMI, or TRAP).
- A *hardware interrupt* is requested by a signal from a physical device. Two types exist:
  - *External* hardware interrupts are triggered by signals at external interrupt pins. All these interrupts are negative-edge triggered and should be active low for at least one CLKOUT1 period to be recognized.
  - *Internal* hardware interrupts are triggered by signals from the on-chip peripherals.

If hardware interrupts are triggered at the same time, the ’C20x services them according to a set priority ranking. Each of the ’C20x interrupts, whether hardware or software, can be placed in one of the following two categories:

- **Maskable interrupts.** These are hardware interrupts that can be blocked (masked) or enabled (unmasked) through software.
- **Nonmaskable interrupts.** These interrupts cannot be blocked. The ’C20x will always acknowledge this type of interrupt and branch from the main program to a subroutine. The ’C20x nonmaskable interrupts include all software interrupts and two external hardware interrupts: reset ($\overline{\text{RS}}$) and $\overline{\text{NMI}}$.

5.6.1 Interrupt Operation: Three Phases

The ’C20x handles interrupts in three main phases:

1) **Receive the interrupt request.** Suspension of the main program must be requested by a software interrupt (from program code) or a hardware interrupt (from a pin or an on-chip device).
2) **Acknowledge the interrupt.** The ’C20x must acknowledge the interrupt request. If the interrupt is maskable, certain conditions must be met in order for the ’C20x to acknowledge it. For nonmaskable hardware interrupts and for software interrupts, acknowledgement is immediate.

3) **Execute the interrupt service routine.** Once the interrupt is acknowledged, the 'C20x branches to its corresponding subroutine called an interrupt service routine (ISR). The 'C20x follows the branch instruction you place at a predetermined address (the vector location) and executes the ISR you have written.

5.6.2 Interrupt Table

For 'C20x devices other than the 'C209, Table 5–5 lists the interrupts available and shows their vector locations. In addition, it shows the priority of each of the hardware interrupts. For the corresponding 'C209 table, see section 11.3, *'C209 Interrupts*, on page 11-10.

Table 5–5. 'C20x Interrupt Locations and Priorities

K†	Vector Location	Name	Priority	Function
0	0h	$\overline{RS}$	1 (highest)	Hardware reset (nonmaskable)
1	2h	$\overline{HOLD}/\overline{INT1}$	4	User-maskable interrupt #1
2	4h	$\overline{INT2}$, $\overline{INT3}$‡	5	User-maskable interrupts #2 and #3
3	6h	TINT	6	User-maskable timer interrupt
4	8h	RINT	7	User-maskable synchronous serial port receive interrupt
5	Ah	XINT	8	User-maskable synchronous serial port transmit interrupt
6	Ch	TXRXINT	9	User-maskable asynchronous serial port transmit/receive interrupt
7	Eh		10	Reserved
8	10h	INT8	–	User-defined software interrupt
9	12h	INT9	–	User-defined software interrupt

Note: This table does not apply to the 'C209. For the 'C209 interrupt table, see section 11.3 on page 11-10.

† The K value is the operand used in an INTR instruction that branches to the corresponding interrupt vector location.

‡ $\overline{INT2}$ and $\overline{INT3}$ have separate pins but are tied to the same vector location.

Table 5–5. ’C20x Interrupt Locations and Priorities (Continued)

K†	Vector Location	Name	Priority	Function
10	14h	INT10	–	User-defined software interrupt
11	16h	INT11	–	User-defined software interrupt
12	18h	INT12	–	User-defined software interrupt
13	1Ah	INT13	–	User-defined software interrupt
14	1Ch	INT14	–	User-defined software interrupt
15	1Eh	INT15	–	User-defined software interrupt
16	20h	INT16	–	User-defined software interrupt
17	22h	TRAP	–	TRAP instruction vector
18	24h	$\overline{\text{NMI}}$	3	Nonmaskable interrupt
19	26h		2	Reserved
20	28h	INT20	–	User-defined software interrupt
21	2Ah	INT21	–	User-defined software interrupt
22	2Ch	INT22	–	User-defined software interrupt
23	2Eh	INT23	–	User-defined software interrupt
24	30h	INT24	–	User-defined software interrupt
25	32h	INT25	–	User-defined software interrupt
26	34h	INT26	–	User-defined software interrupt
27	36h	INT27	–	User-defined software interrupt
28	38h	INT28	–	User-defined software interrupt
29	3Ah	INT29	–	User-defined software interrupt
30	3Ch	INT30	–	User-defined software interrupt
31	3Eh	INT31	–	User-defined software interrupt

Note: This table does not apply to the ’C209. For the ’C209 interrupt table, see section 11.3 on page 11-10.

† The K value is the operand used in an INTR instruction that branches to the corresponding interrupt vector location.

‡ $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ have separate pins but are tied to the same vector location.

5.6.3 Maskable Interrupts

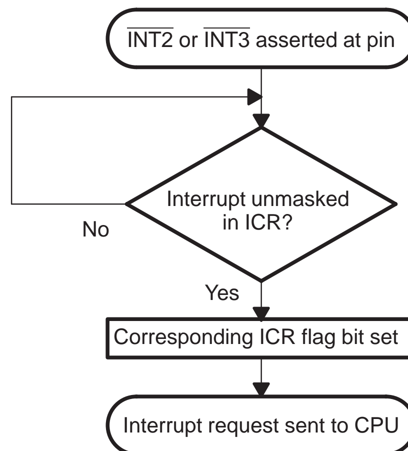
When a maskable interrupt is successfully requested by a hardware device or by an external pin, the corresponding flag or flags are activated. These flags are activated whether or not the interrupt is later acknowledged by the processor.

Two registers on the 'C20x contain flag bits:

- Interrupt flag register (IFR), a 16-bit, memory-mapped register located at address 0006h in data-memory space.The IFR is explained in detail in section 5.6.4
- Interrupt control register (ICR), a 16-bit register located at address FFECh in I/O space.The ICR is explained in section 5.6.6.

The IFR contains flag bits for all the maskable interrupts. The ICR contains additional flag bits for the interrupts INT2 and INT3. For all maskable interrupts except INT2 and INT3, an interrupt request is sent to the CPU as soon as the interrupt signal is sent by the pin or on-chip peripheral. For INT2 or INT3, the interrupt request is only sent to the CPU if the interrupt signal is not masked by its mask bit in the ICR. Figure 5–5 shows the process for successfully requesting INT2 or INT3.

Figure 5–5. INT2/INT3 Request Flow Chart



After an interrupt request is received by the CPU, the CPU must decide whether to acknowledge the request. Maskable hardware interrupts are acknowledged only after certain conditions are met:

- **Priority is highest.** When more than one hardware interrupt is requested at the same time, the ’C20x services them according to a set priority ranking in which 1 indicates the highest priority. For the priorities of the hardware interrupts, see section 5.6.2 (on page 5-16).
- **IMR mask bit is 1.** The interrupt must be unmasked (enabled) in the interrupt mask register (IMR), a 16-bit, memory-mapped register located at address 0004h in data-memory space. The IMR contains mask bits for all the maskable interrupts. $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ share one of the bits in the IMR. The IMR is explained in section 5.6.5 on page 5-23.
- **INTM bit is 0.** The interrupt mode (INTM) bit, bit 9 of status register ST0, enables or disables all maskable interrupts:

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

  INTM is set to 1 automatically when the CPU acknowledges an interrupt (except when initiated by the TRAP instruction). INTM can also be set to 1 by a hardware reset or by execution of a disable-interrupts instruction (SETC INTM). You can clear INTM by executing the enable-interrupts instruction (CLRC INTM). INTM has no effect on reset, $\overline{\text{NMI}}$, or software-interrupts (initiated with the TRAP, NMI, and INTR instructions). Also, INTM is unaffected by the LST (load status register) instruction.

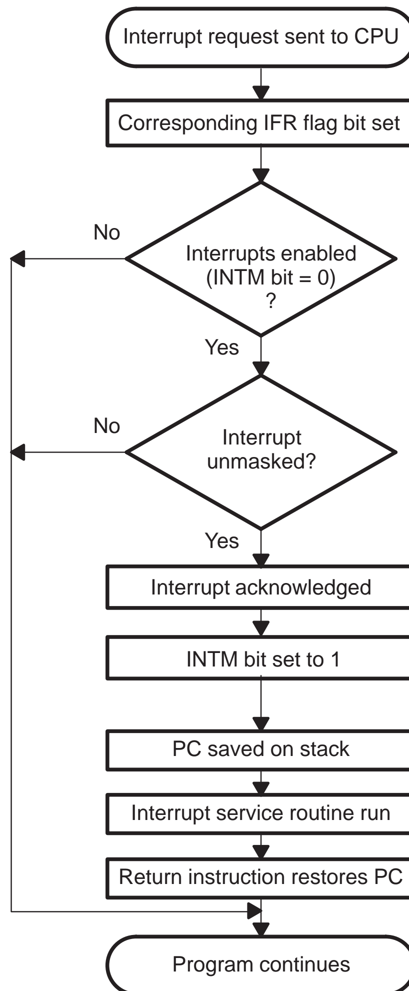
  INTM does not modify the interrupt flag register (IFR), the interrupt mask register (IMR), or the interrupt control register (ICR).

When the CPU acknowledges a maskable hardware interrupt, it loads the instruction bus with the INTR instruction. This instruction forces the CPU to branch to the corresponding *interrupt vector location*. From this location in program memory, the CPU fetches a branch that leads to the appropriate interrupt service routine. As the CPU branches to the interrupt service routine, it also sets the INTM bit to 1, preventing all hardware-initiated maskable interrupts from interrupting the execution of the ISR. Note that the INTR instruction can also be initiated directly by software; thus, the interrupt service routines for the maskable interrupts can also be initiated directly with the INTR instruction (see section 5.6.7, *Nonmaskable Interrupts* on page 5-27).

To determine which vector address has been assigned to each of the interrupts, see section 5.6.2 (on page 5-16). Interrupt vector locations are spaced apart by two addresses so a 2-word branch instruction can be accommodated in each of the locations.

Figure 5–6 summarizes how maskable interrupts are handled by the CPU.

Figure 5–6. Maskable Interrupt Operation Flow Chart



5.6.4 Interrupt Flag Register (IFR)

The 16-bit interrupt flag register (IFR), located at address 0006h in data memory space, contains flag bits for all the maskable interrupts. When a maskable interrupt request reaches the CPU, the corresponding flag is set to 1 in the IFR. This indicates that the interrupt is pending, or waiting for acknowledgement.

Read the IFR to identify pending interrupts, and write to the IFR to clear pending interrupts. To clear an interrupt request (and set its IFR flag to 0), write

a 1 to the corresponding IFR bit. All pending interrupts can be cleared by writing the current contents of the IFR back into the IFR. Acknowledgement of a hardware request also clears the corresponding IFR bit. A device reset clears all IFR bits.

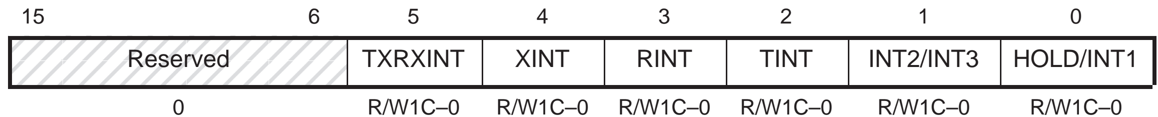
Notes:

1) When an interrupt is requested by an INTR instruction, if the corresponding IFR bit is set, the CPU will not clear it automatically. If an application requires that the IFR bit be cleared, the bit must be cleared in the interrupt service routine.

2) To avoid double interrupts from the synchronous serial port and the asynchronous serial port (including delta interrupts), clear the IFR bit(s) in the corresponding interrupt service routine, just before returning from the routine.

For ’C20x devices other than the ’C209, Figure 5–7 shows the IFR. Descriptions of the bits follow the figure. For a description of the ’C209 IFR, see section 11.3.1, *’C209 Interrupt Registers*, on page 11-12.

Figure 5–7. ’C20x Interrupt Flag Register (IFR) — Data-Memory Address 0006h



Note: 0 = Always read as zeros; R = Read access; W1C = Write 1 to this bit to clear it to 0; value following dash (–) is value after reset.

Table 5–6. ’C20x IFR — Data-Memory Address 0006h Bit Descriptions

Bit No.	Name	Function
15–6	Reserved	Bits 15–6 are reserved and are always read as 0s.
5	TXRXINT	Transmit/receive interrupt flag. Bit 5 is tied to the transmit/receive interrupt for the asynchronous serial port. *To avoid double interrupts, write a 1 to this bit in the interrupt service routine.*
		0 Interrupt TXRXINT is not pending.
		1 Interrupt TXRXINT is pending.

Table 5–6. ’C20x IFR — Data-Memory Address 0006h Bit Descriptions (Continued)

Bit No.	Name	Function
4	XINT	Transmit interrupt flag. Bit 4 is tied to the transmit interrupt for the synchronous serial port. *To avoid double interrupts, write a 1 to this bit in the interrupt service routine.* 0 Interrupt XINT is not pending. 1 Interrupt XINT is pending.
3	RINT	Receive interrupt flag. Bit 3 is tied to the receive interrupt for the synchronous serial port. *To avoid double interrupts, write a 1 to this bit in the interrupt service routine.* 0 Interrupt RINT is not pending. 1 Interrupt RINT is pending.
2	TINT	Timer interrupt flag. Bit 2 is tied to the timer interrupt, TINT. 0 Interrupt TINT is not pending. 1 Interrupt TINT is pending.
1	INT2/INT3	Interrupt 2/Interrupt 3 flag. The $\overline{\text{INT2}}$ pin and the $\overline{\text{INT3}}$ pin are both tied to bit 1. If $\overline{\text{INT2}}$ is requested, INT2/INT3 and FINT2 of the interrupt control register (ICR) are both automatically set to 1. If $\overline{\text{INT3}}$ is requested, INT2/INT3 and FINT3 (of the ICR) are both automatically set to 1. 0 Neither $\overline{\text{INT2}}$ nor $\overline{\text{INT3}}$ is pending. 1 At least one of the two interrupts is pending. To determine which one is pending or if both are pending, read flag bits FINT2 and FINT3 in the ICR. FINT2 and FINT3 are not automatically cleared when $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ are acknowledged by the CPU; they must be cleared by the interrupt service routine.
0	HOLD/INT1	HOLD/Interrupt 1 flag. Bit 0 is a flag for $\overline{\text{HOLD}}$ or $\overline{\text{INT1}}$. The operation of the $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ pin differs depending on the value of the MODE bit in the ICR. When MODE = 1, an interrupt is triggered only by a negative edge on the pin. When MODE = 0, interrupts can be triggered by both a negative edge and a positive edge. This is necessary to implement the ’C20x HOLD operation (see section 4.6, *Direct Memory Access Using The HOLD Operation*, on page 4-18). 0 $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ is not pending. 1 $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ is pending.

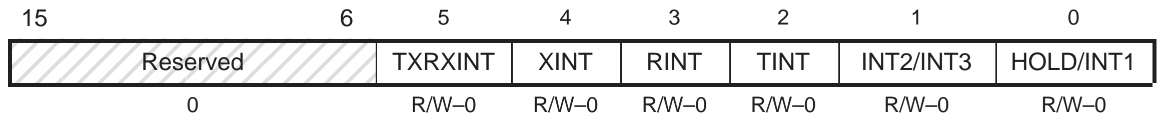
5.6.5 Interrupt Mask Register (IMR)

The 16-bit interrupt mask register (IMR), located at address 0004h in data-memory space, is used for masking external and internal hardware interrupts. Neither $\overline{\text{NMI}}$ nor $\overline{\text{RS}}$ is included in the IMR; thus, IMR has no effect on these interrupts.

Read the IMR to identify masked or unmasked interrupts, and write to the IMR to mask or unmask interrupts. To unmask an interrupt, set its corresponding IMR bit to 1. To mask an interrupt, set its corresponding IMR bit to 0. The IMR bits are not affected by a device reset.

For 'C20x devices other than the 'C209, Figure 5–8 shows the IMR. Descriptions of the bits follow the figure. For a description of the 'C209 IMR, see section 11.3.1, *'C209 Interrupt Registers*, on page 11-12.

Figure 5–8. 'C20x Interrupt Mask Register (IMR) — Data-Memory Address 0004h



Note: 0 = Always read as zeros; R = Read access; W = Write access; value following dash (–) is value after reset.

Table 5–7. 'C20x IMR — Data-Memory Address 0004h Bit Descriptions

Bit No.	Name	Function
15–6	Reserved	Bits 15–6 are reserved and are always read as 0s.
5	TXRXINT	Transmit/receive interrupt mask. Bit 5 is tied to the transmit/receive interrupt for the asynchronous serial port. 0 Interrupt TXRXINT is masked. 1 Interrupt TXRXINT is unmasked.
4	XINT	Transmit interrupt mask. Bit 4 is tied to the transmit interrupt for the synchronous serial port. 0 Interrupt XINT is masked. 1 Interrupt XINT is unmasked.
3	RINT	Receive interrupt mask. Bit 3 is tied to the receive interrupt for the synchronous serial port. 0 Interrupt RINT is masked. 1 Interrupt RINT is unmasked.

Table 5–7. ’C20x IMR — Data-Memory Address 0004h Bit Descriptions (Continued)

Bit No.	Name	Function
2	TINT	Timer interrupt mask. Bit 2 is tied to the interrupt for the timer. 0 Interrupt TINT is masked. 1 Interrupt TINT is unmasked.
1	INT2/INT3	Interrupt 2/Interrupt 3 mask. The $\overline{\text{INT2}}$ pin and the $\overline{\text{INT3}}$ pin are both tied to bit 1. With this bit, you mask both $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ simultaneously. In conjunction with this bit, bits MINT2 and MINT3 of the ICR are used to individually unmask $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$. 0 $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ are masked. 1 If INT2/INT3 = 1 and MINT2 = 1, $\overline{\text{INT2}}$ is unmasked. If INT2/INT3 = 1 and MINT3 = 1, $\overline{\text{INT3}}$ is unmasked.
0	HOLD/INT1	HOLD/Interrupt 1 mask. This bit masks or unmasks interrupts requested at the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin. 0 $\overline{\text{HOLD}}/\overline{\text{INT1}}$ is masked. 1 $\overline{\text{HOLD}}/\overline{\text{INT1}}$ is unmasked.

5.6.6 Interrupt Control Register (ICR)

The 16-bit interrupt control register (ICR), located at address FFECh in I/O space, controls the function of the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin and individually controls the interrupts $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$.

Controlling the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin

This pin can be used for triggering the interrupt $\overline{\text{INT1}}$ and for sending a $\overline{\text{HOLD}}$ signal to the CPU. Accordingly, the MODE bit provides two possible modes for the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin. When MODE = 1, the pin is negative-edge sensitive and, thus, is set appropriately for initiating a standard interrupt ($\overline{\text{INT1}}$). When MODE = 0, the pin is both negative- and positive-edge sensitive, which is necessary for implementing the logic for the HOLD operation (see section 4.6, *Direct Memory Access Using The HOLD Operation*, on page 4-18). Regardless of the value of MODE, the pin is connected to the same interrupt logic, which initiates only one interrupt service routine. ($\overline{\text{HOLD}}/\overline{\text{INT1}}$ is mapped to interrupt vector location 0002h in program memory.) To differentiate the two uses of the pin, the interrupt service routine must test the value of the MODE bit.

Controlling $\overline{INT2}$ and $\overline{INT3}$

Each of these interrupts has its own pin. However, they share:

- A single flag bit (INT2/INT3) in the interrupt flag register (IFR).
- A single mask bit in the interrupt mask register (IMR).
- A single interrupt service routine. ($\overline{INT2}$ and $\overline{INT3}$ are mapped to interrupt vector location 0004h in program memory.)

To allow you to use $\overline{INT2}$ and $\overline{INT3}$ individually, the ICR provides two mask bits (MINT2 and MINT3) and two flag bits (FINT2 and FINT3).

When interrupts are requested on the pins $\overline{INT2}$ and $\overline{INT3}$, MINT2 and MINT3 determine whether the flag bits FINT2, FINT3, and INT2/INT3 are set. To mask $\overline{INT2}$ (prevent the setting of flags FINT2 and INT2/INT3), write a 0 to MINT2; to mask $\overline{INT3}$ (prevent the setting of flags FINT3 and INT2/INT3) write a 0 to MINT3. If INT2/INT3 is not set, the CPU has not received and will not acknowledge the interrupt request.

When INT2/INT3 is set, one or both of the interrupts is pending. To differentiate the occurrences of the two interrupts, your interrupt service routine can test FINT2 and FINT3 and then branch to the appropriate subroutine. If you want the interrupt service routine to be executed only in response to one of the interrupts, mask the other interrupt in the ICR. Each of the ICR flag bits, like the IFR flag bit, can be cleared by writing a 1 to it.

Note:

1) Neither FINT2 nor FINT3 is automatically cleared when the CPU acknowledges the corresponding interrupt. If the application requires the bit(s) be cleared, the clearing must be done in the interrupt service routine.

2) Writing 1s to FINT2 and FINT3 will set these bits to 0 but will *not* clear interrupt requests for $\overline{INT2}$ and $\overline{INT3}$. To clear requests for $\overline{INT2}$ and/or $\overline{INT3}$, write a 1 to the INT2/INT3 bit of the IFR.

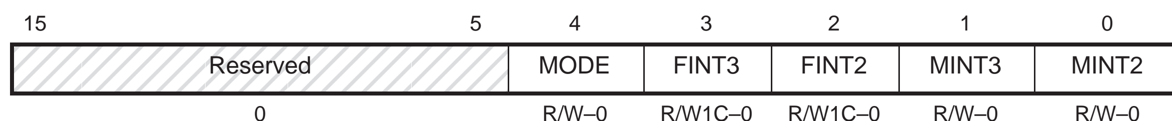
If INT2 or INT3 is unmasked in the ICR, the IFR flag bit will be set regardless of bit 1 (INT2/INT3) in the IMR. If the IFR flag bit is set, the IMR bit is set, and the INTM bit is 0 (maskable interrupts are enabled), the CPU will acknowledge the interrupt. If an interrupt is masked by the IMR and/or the ICR, it will not be acknowledged, even if INTM = 0.

At reset, all ICR bits are set to zero, which means:

- The $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin is both negative- and positive-edge sensitive (MODE = 0).
- The FINT2 and FINT3 flag bits are cleared.
- $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ are masked.

Figure 5–9 shows the ICR, and bit descriptions follow the figure.

Figure 5–9. ’C20x Interrupt Control Register (ICR) — I/O-Space Address FFECh



Note: 0 = Always read as zeros; R = Read access; W = Write access; W1C = Write 1 to this bit to clear it to 0; value following dash (–) is value after reset.

Table 5–8. ’C20x ICR — I/O-Space Address FFECh Bit Descriptions

Bit No.	Name	Function
15–5	Reserved	Bits 15–5 are reserved and are always read as 0s.
4	Mode	Pin mode. Bit 4 selects one of two possible modes for the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin.
		0 *Double-edge mode*. The $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin is both negative- and positive-edge sensitive. A falling edge or a rising edge triggers an interrupt request. This mode is necessary for proper implementation of a HOLD operation.
		1 *Single-edge mode*. A falling edge (only) on the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin triggers an interrupt request.
3	FINT3	Interrupt 3 flag. If MINT3 = 1, an interrupt request on the $\overline{\text{INT3}}$ pin sets FINT3 and bit 1 of the IFR (INT2/INT3).
		0 $\overline{\text{INT3}}$ is not pending.
		1 $\overline{\text{INT3}}$ is pending.
2	FINT2	Interrupt 2 flag. If MINT2 = 1, an interrupt request on the $\overline{\text{INT2}}$ pin sets FINT2 and bit 1 of the IFR (INT2/INT3).
		0 $\overline{\text{INT2}}$ is not pending.
		1 $\overline{\text{INT2}}$ is pending.

Table 5–8. ’C20x ICR — I/O-Space Address FFECh Bit Descriptions (Continued)

Bit No.	Name	Function
1	MINT3	Interrupt 3 mask. This bit masks the external interrupt $\overline{\text{INT3}}$ or, in conjunction with the INT2/INT3 bit of the IMR, unmasks $\overline{\text{INT3}}$. 0 $\overline{\text{INT3}}$ is masked. Neither FINT3 nor bit 1 of the IFR (INT2/INT3) is set by a request on the $\overline{\text{INT3}}$ pin. 1 $\overline{\text{INT3}}$ is unmasked. Flag bits FINT3 and INT2/INT3 are both set by a request on the $\overline{\text{INT3}}$ pin.
0	MINT2	Interrupt 2 mask. This bit masks the external interrupt $\overline{\text{INT2}}$ or, in conjunction with the INT2/INT3 bit of the IMR, unmasks $\overline{\text{INT2}}$. 0 $\overline{\text{INT2}}$ is masked. Neither FINT2 nor bit 1 of the IFR (INT2/INT3) is set by a request on the $\overline{\text{INT2}}$ pin. 1 $\overline{\text{INT3}}$ is unmasked. Flag bits FINT2 and INT2/INT3 are both set by a request on the $\overline{\text{INT2}}$ pin.

5.6.7 Nonmaskable Interrupts

Hardware nonmaskable interrupts can be requested through two pins:

- **$\overline{\text{RS}}$ (reset).** $\overline{\text{RS}}$ is an interrupt that stops program flow, returns the processor to a predetermined state, and then begins program execution at address 0000h. For details of the reset operation, see section 5.7, *Reset Operation*, on page 5-35. When $\overline{\text{RS}}$ is acknowledged, the interrupt mode (INTM) bit of status register ST1 is set to 1 to disable maskable interrupts.

- **$\overline{\text{NMI}}$**. When $\overline{\text{NMI}}$ is activated (either by the $\overline{\text{NMI}}$ pin or by the NMI instruction), the processor switches program control to vector location 24h. In addition, maskable interrupts are disabled (the INTM bit of status register ST0 is set to 1). Although $\overline{\text{NMI}}$ uses the same logic as the maskable interrupts, it is not maskable. $\overline{\text{NMI}}$ happens regardless of the value of the INTM bit, and no mask bit exists for $\overline{\text{NMI}}$. If the $\overline{\text{NMI}}$ pin is not used, it should be pulled high to prevent an accidental interrupt.

  $\overline{\text{NMI}}$ can be used as a soft reset. Unlike a hardware reset ($\overline{\text{RS}}$), the $\overline{\text{NMI}}$ neither affects any of the modes of the device nor aborts a currently active instruction or memory operation.

Software interrupts (which are inherently nonmaskable) are requested by the following instructions:

- **INTR.** This instruction allows you to initiate any ’C20x interrupt, including user-defined interrupts INT8 through INT16 and INT20 through INT31.

The instruction operand (K) indicates which interrupt vector location the CPU will branch to. To determine the operand K that corresponds to each interrupt vector location see section 5.6.2 (on page 5-16). When an INTR interrupt is acknowledged, the interrupt mode (INTM) bit of status register ST1 is set to 1 to disable maskable interrupts.

Note:

The INTR instruction does not affect IFR flags. When you use the INTR instruction to initiate an interrupt that has an associated flag bit in the IFR, the instruction neither sets nor clears the flag bit. No software write operation can set the IFR flag bits; only the appropriate hardware requests can. If a hardware request has set the flag for an interrupt and then the INTR instruction is used to initiate that interrupt, the INTR instruction will not clear the flag.

- ❑ **NMI.** This instruction forces a branch to interrupt vector location 24h, the same location used for the nonmaskable hardware interrupt $\overline{\text{NMI}}$. Thus, you can either initiate $\overline{\text{NMI}}$ by driving the $\overline{\text{NMI}}$ pin low or by executing an NMI instruction. When the NMI instruction is executed, INTM is set to 1 to disable maskable interrupts.

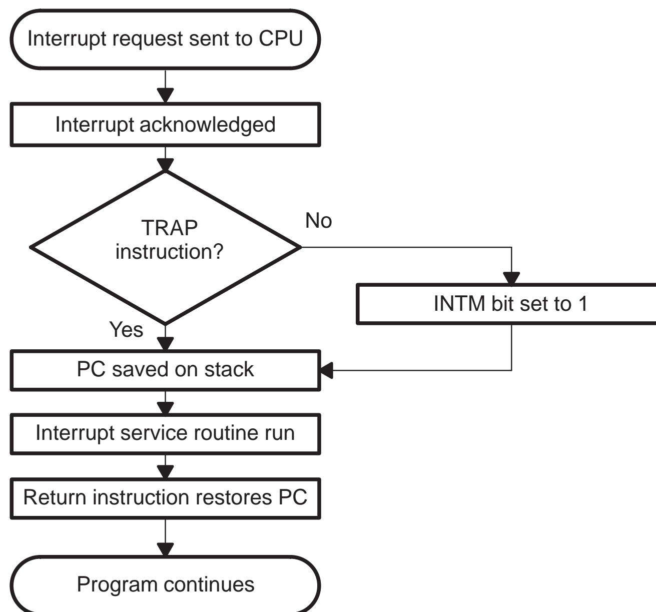
- ❑ **TRAP.** This instruction forces the CPU to branch to interrupt vector location 22h. The TRAP instruction does *not* disable maskable interrupts (INTM is not set to 1); thus when the CPU branches to the interrupt service routine, that routine can be interrupted by the maskable hardware interrupts (in addition to $\overline{\text{RS}}$ and $\overline{\text{NMI}}$).

If the INTM bit is set to 1 during the acknowledgement process, all hardware-initiated maskable interrupts are disabled and, thus, cannot interfere with the interrupt service routine.

To determine which vector address has been assigned to each of the interrupts on a specific ’C20x device, see section 5.6.2 (on page 5-16). Interrupt vector locations are spaced apart by two addresses so that a 2-word branch instruction can be accommodated in each location.

Figure 5–10 summarizes how nonmaskable interrupts are handled by the CPU.

Figure 5–10. Nonmaskable Interrupt Operation Flow Chart



5.6.8 Interrupt Service Routines (ISRs)

After an interrupt has been requested and acknowledged, the CPU follows an interrupt vector to the ISR. The ISR is the program code that actually performs the tasks requested by the interrupt. While performing these tasks, the ISR may also be:

- ❑ Saving and restoring register values
- ❑ Managing ISRs within ISRs

Saving and restoring register values

Only the incremented program counter value is stored automatically before the CPU enters an interrupt service routine (ISR). You must design the ISR to save and then restore any other important register values. For example, if your ISR will need to perform a multiplication, it will need to use the product register (PREG). If the value currently in the PREG must be in the PREG after the ISR, the ISR must save the value, perform the new multiplication, store the resulting PREG value, and then reload the original value. You may find that certain registers will need to be saved during most ISRs. If so, you can copy a common save and restore routine and then individualize it for each interrupt.

Managing ISRs within ISRs

The ’C20x hardware stack allows you to have ISRs within ISRs. When considering nesting ISRs like this, keep the following in mind:

- If you want the ISR be interrupted by a maskable interrupt, the ISR must unmask the interrupt by setting the appropriate IMR bit (and ICR bit, if applicable) and executing the enable-interrupts instruction (CLRC INTM).
- The hardware stack is limited to eight levels. Each time an interrupt is serviced or a subroutine is entered, the return address is pushed onto the hardware stack. This provides a way to return to the previous context afterwards. The stack contains eight locations, allowing interrupts or subroutines to be nested up to eight levels deep. (One level of the stack is reserved for debugging, to be used for breakpoint/single-step operations. If debugging is not used, this extra level is available for internal use.) If your software requires more than eight stack levels, you can use the POPD and PSHD instructions to effectively extend the stack into data memory.
- If you do not nest ISRs, you can avoid stack overflow. The ’C20x has a feature that allows you to prevent unintentional nesting. If an interrupt occurs during the execution of a CLRC INTM instruction, the device always completes CLRC INTM as well as the next instruction before the pending interrupt is processed. This ensures that a return instruction that directly follows CLRC INTM will be executed before an interrupt is processed. The return instruction will pop the previous return address off the top of the stack before the new return address is pushed onto the stack.

  To allow the CPU to complete the return, interrupts are also blocked after a RET instruction until at least one instruction at the return address is executed. Interrupts may be blocked for more than one instruction if the instruction at the return address requires additional blocking for pipeline protection.
- If you want an ISR to occur *within* the current ISR rather than *after* the current ISR, place the CLRC INTM instruction more than one instruction before the return (RET) instruction.

5.6.9 Interrupt Latency

The length of an interrupt latency—the delay between when an interrupt request is made and when it is serviced—depends on many factors. For example, the CPU always completes all instructions in the pipeline before executing a software vector. This section describes the factors that determine minimum latency and then describes factors that may cause additional latency. The maximum latency is a function of wait states and pipeline protection.

For an external, maskable hardware interrupt, a minimum latency of eight cycles is required to synchronize the interrupt externally, recognize the interrupt, and branch to the interrupt vector location. On the ninth cycle, the interrupt vector is fetched. For a software interrupt, the minimum latency consists of four cycles needed to branch to the interrupt vector location.

Latency for pipeline protection

Multicycle instructions add additional cycles to empty the pipeline. Instructions may become multicycle for these reasons:

- An instruction that writes to or reads from external memory may be delayed by wait states generated by the external READY pin or the on-chip wait-state generator. These wait states may affect the instruction being executed at the time the interrupt is requested, and they may affect the interrupt itself if the interrupt vector must be fetched from external memory.
- If an interrupt occurs during a HOLD operation and the interrupt vector must be fetched from external memory, the vector cannot be fetched until $\overline{\text{HOLDA}}$ is deasserted.
- When repeated with RPT, instructions run parallel operations in the pipeline and the context of these additional parallel operations cannot be saved in an interrupt service routine. To protect the context of the repeated instruction, the CPU locks out all interrupts except reset until the RPT loop completes.

Note:

Reset ($\overline{\text{RS}}$) is not delayed by multicycle instructions. $\overline{\text{NMI}}$ can be delayed by multicycle instructions.

Latency for stack overflow protection

A return address (incremented program counter value) is forced onto the hardware stack every time the CPU follows another interrupt service routine or other subroutine. However, the ’C20x has a feature that can help you to keep the hardware stack from overflowing. Interrupts cannot be processed between the CLRC INTM (enable maskable interrupts) instruction and the next instruction in a program sequence. This ensures that a return instruction that directly follows CLRC INTM will be executed before an interrupt is processed. The return instruction will pop the previous return address off the top of the stack before the new return address is pushed onto the stack. If the interrupt were to occur before the return, the new return address would be added to the hardware stack, even if the stack were already full.

To allow the CPU to complete the return, interrupts are also blocked after a RET instruction until at least one instruction at the return address is executed.

5.6.10 Context Saving During Interrupts

During context saving and restoring, the order in which registers ST0 and ST1 are loaded is crucial and changes contingent upon the addressing mode (direct and indirect). As there is no LPL instruction, you can extend interruptability by:

- ❑ Direct addressing context save
- ❑ Indirect addressing context save (software stack)

See Figure 5–11 and Figure 5–12 for code examples.

- ❑ Direct addressing context save

  Using direct addressing to perform context save to data memory is the simplest way to extend interruptability to the second level of depth. The code example below shows the most likely items to be saved, and in so doing, demonstrates most of the techniques used for contexting in general. Note, however, that this is not a comprehensive context save operation, and that you must consider which registers will, and will not, be maintained for the specific ISR. Given the large number of registers present on the 'C20x, it is not recommended that you employ a generic, all encompassing context save process, as this would almost always be impractical.

Figure 5–11. Direct Addressing Context Save

```
STATUS  .usect  "BLOCKB2", 2        ; Must be located on Data Page 0
        .bss    CONTEX, 4, 1        ; Located anywhere in Data Memory
        .text
ISR1:   SST     #0,STATUS           ; ST0 must go to data page 0
        SST     #1,STATUS+1         ; ST1 must go to data page 0
        LDP     #CONTEX             ;
        SACH    CONTEX              ; Save ACCH & ACCL
        SACL    CONTEX+1            ; (if needed, P & T regs saved as shown above)
        POPD    CONTEX+2            ; Offload 1 level of stack
        BLDD    #04h, CONTEX+3      ; Save IMR
        LDP     #0
        LACL    #0010B              ; Mask to sub-enable only INT2, for example
        SACL    04h                 ; Write to IMR
        CLRC    INTM                ; Re-allow interruptability
               *                    ;
               *                    ; Nestable ISR goes here. . .
               *                    ;
        SETC    INTM                ; Interruptability back off
        LDP     #CONTEX             ; Go to page with context values
        PSHD    CONTEX+2            ; Reload stack with return address
        LACL    CONTEX+1            ; Restore ACCL w/o sign extension
        ADD     CONTEX,16           ; Sum in ACCH
        LDP     #0                  ; Go to DP=0. for status registers
        BLDD    #CONTEX+3, 04h      ; Restore to IMR
        LST     #1, STATUS+1        ; Restore ST1
        LST     #0, STATUS          ; Restore ST0
        CLRC    INTM                ; Enable interrupts
        RET                         ;
```

- ❑ Indirect addressing context save (software stack)

  Using indirect addressing to perform a context save allows any degree of nestability of interrupts and is typically used in conjunction with a software stack. In creating a software stack, you should assign one auxiliary register (AR) as a stack pointer. Following TI's C compiler convention, AR1 has been assigned as the stack pointer (SP).

Figure 5–12. Indirect Addressing Context Save

```
        .bss    STACK,100h      ; Assign 512 locations for stack
        .text
OSR1:   LAR     AR1, #STACK     ; AR1 is SP, start at beginning
        *
        *
ISR1:   MAR     *, AR1          ; Select AR1 to point to stack
        SST     #1,*+           ; Save ST1 & ST0
        SST     #0,*+
        SACH    *+              ; Save ACCH & ACCL
        SACL    *+
        LDP     #0
        LACC    4h              ; Get IMR
        SACL    *+              ; Store old IMR
        POPD    *+              ; Offload 1 level of stack
        LACL    #010B           ; Mask to sub-enable only INT2
        SACL    4h              ; New IMR
        CLRC    INTM            ; Re-allow interruptability
               *                ;
               *                ; Interruptible ISR goes here
               *                ;
        SETC    INTM            ; Interruptability back off
        MAR     *,AR1           ; Select stack pointer
        MAR     *-              ; Move AR1 to last saved content
        PSHD    *-              ; Reload stack with return address
        LACC    *-              ; Get & restore original IMR value
        LDP     #0
        SACL    4h              ; Restore IMR
        LACL    *-              ; Load ACCL & sum in ACCH
        ADD     *-,16
        LST     #0,*-           ; Restore ST0
        LST     #1,*            ; Restore ST1 and ARP
        CLRC    INTM            ; Enable interrupts
        RET                     ; Return to main
```

5.7 Reset Operation

Reset ($\overline{RS}$) is a nonmaskable external interrupt that can be used at any time to put the 'C20x into a known state. Reset is the highest priority interrupt; no other interrupt takes precedence over reset. Reset is typically applied after power up when the machine is in an unknown state. Because the reset signal aborts memory operations and initializes status bits, the system should be reinitialized after each reset. The $\overline{NMI}$ interrupt can be used for soft resets because it neither aborts memory operations nor initializes status bits.

Driving $\overline{RS}$ low causes the 'C20x to terminate execution and affects various registers and status bits. For correct system operation after power up, $\overline{RS}$ must be asserted for at least six clock cycles. The device latches the reset pulse and generates an internal reset pulse long enough to ensure a device reset. The device fetches its first instruction 16 cycles after the rising edge of $\overline{RS}$. Processor execution begins at location 0000h, which normally contains a branch instruction to the system initialization routine.

When the 'C20x receives a reset signal, the following actions take place:

- **Control features:**
  - The program counter is cleared to 0 (however, the address bus, A15–A0, is unknown while $\overline{RS}$ is low).
  - Status bits in registers ST0 and ST1 are loaded with their reset values: OV = 0, INTM = 1, CNF = 0, SXM = 1, C = 1, XF= 1 and PM = 00. (The other status bits remain undefined and should be initialized by a reset.)
  - The INTM (interrupt mode) bit is set to 1, disabling all maskable interrupts. ($\overline{RS}$ and $\overline{NMI}$ are not maskable.) Also, the interrupt flag register (IFR), interrupt mask register (IMR), and interrupt control register (ICR) are cleared.
  - The MODE bit of the interrupt control register (ICR) is set to 0 so that the $\overline{HOLD}/\overline{INT1}$ pin is both negative- and positive-edge sensitive.
  - The repeat counter (RPTC) is cleared.
- **Memory and I/O spaces:**
  - A logic 0 is loaded into the CNF (configuration control) bit in status register ST1, mapping dual-access RAM block B0 into data space.
  - The global memory allocation register (GREG) is cleared to make all memory local.
  - The wait-state generator is set to provide the maximum number of wait states for external memory and I/O accesses.

- **Peripherals:**

The peripherals are not reset until 16 CLKOUT1 cycles from the rising edge of the RESET pin.

- The timer count is set to its maximum value (FFFFh), the timer divide-down value is set to 0, and the timer starts counting down.
- The synchronous serial port is reset:
  - The port emulation mode is set to immediate stop.
  - Error and status flags are reset.
  - Receive interrupts are set to occur when the receive buffer is not empty.
  - Transmit interrupts are set to occur when the transmit buffer can accept one or more words.
  - External clock and frame synchronization sources are selected.
  - Continuous mode is selected.
  - Digital loopback mode is disabled.
  - The receiver and transmitter are enabled.
- The asynchronous serial port is reset:
  - The port emulation mode is set to immediate stop.
  - Error and status flags are reset.
  - Receive, transmit, and delta interrupts are disabled.
  - One stop bit is selected.
  - Auto-baud alignment is disabled.
  - The TX pin is forced high between transmissions.
  - I/O pins IO0, IO1, IO2, and IO3 are configured as inputs.
  - A baud rate of (CLKOUT1 rate)/16 is selected.
  - The port is disabled.
- CLK register bit 0 is cleared to 0 so that the CLKOUT1 signal is available at the CLKOUT1 pin.

No other registers or status bits (such as the accumulator, DP, ARP, and the auxiliary registers) are initialized. Table 5–9 and Table 5–10 list the reset values for all the registers mapped to on-chip addresses.

Table 5–9. Reset Values of On-Chip Registers Mapped to Data Space

Name	Data-Memory Address	Reset Value	Description
IMR	0004h	0000h	Interrupt mask register
GREG	0005h	0000h	Global memory allocation register
IFR	0006h	0000h	Interrupt flag register

Table 5–10. Reset Values of On-Chip Registers Mapped to I/O Space

	I/O Address			
Name	’C209	Other ’C20x	Reset Value	Description
PMST	–	FFE4h	0000x	Program memory status register
CLK	–	FFE8h	0000h	CLKOUT1-pin control (CLK) register
ICR	–	FFECh	0000h	Interrupt control register
SDTR	–	FFF0h	xxxxh	Synchronous data transmit and receive register
SSPCR	–	FFF1h	0030h	Synchronous serial port control register
SSPST	–	FFF2h	0000h	Synchronous serial port status register
SSPMC	–	FFF3h	0000h	Synchronous serial port multichannel register
ADTR	–	FFF4h	xxxxh	Asynchronous data transmit and receive register
ASPCR	–	FFF5h	0000h	Asynchronous serial port control register
IOSR	–	FFF6h	18xxh	I/O status register
BRD	–	FFF7h	0001h	Baud-rate divisor register
TCR	FFFCh	FFF8h	0000h	Timer control register
PRD	FFFDh	FFF9h	FFFFh	Timer period register
TIM	FFFEh	FFFAh	FFFFh	Timer counter register
SSPCT	–	FFFBh	0000h	Synchronous serial port shift clock and frame sync prescaler
WSGR	FFFFh	FFFCh	0FFFh	Wait-state generator control register

Note: An x in an address represents four bits that are either not affected by reset or dependent on pin levels at reset.

5.7.1 TMS320C206/LC206 Reset and PLL Lock Conditions

TMS320C206/LC206 devices have special reset conditions compared to the TMS320C203 and TMS320F206 devices. Table 5–11 explains the reset conditions for the TMS320C206/LC206 devices.

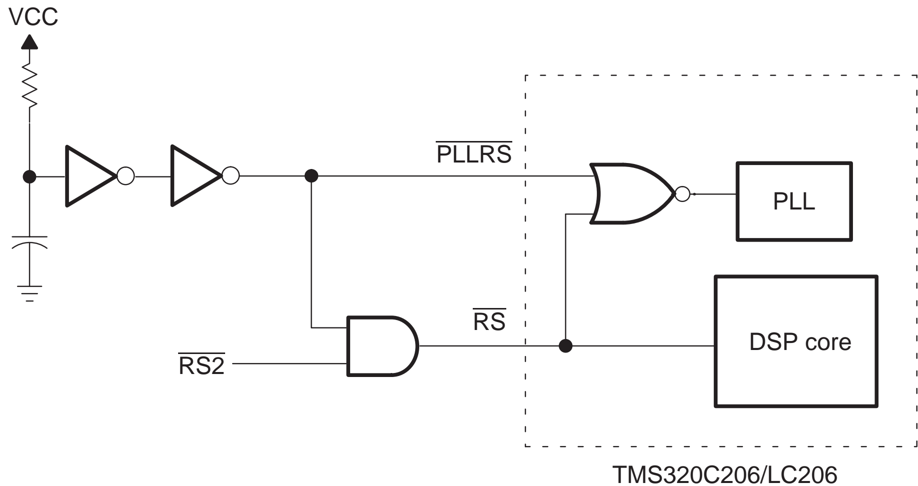
Table 5–11. Reset Conditions for the 'C206/'LC206

Condition	$\overline{\text{PLLRS}}$	$\overline{\text{RS2}}$	$\overline{\text{RS}}$	PLL †	DSP Core
Power on reset (POR)	0	X (Don't care)	0	Reset	Reset
After POR	Always 1	1	1	No	No
After POR	Always 1	0	0	No	Reset

† PLL-reset means that the PLL resets and initiates locking sequence.

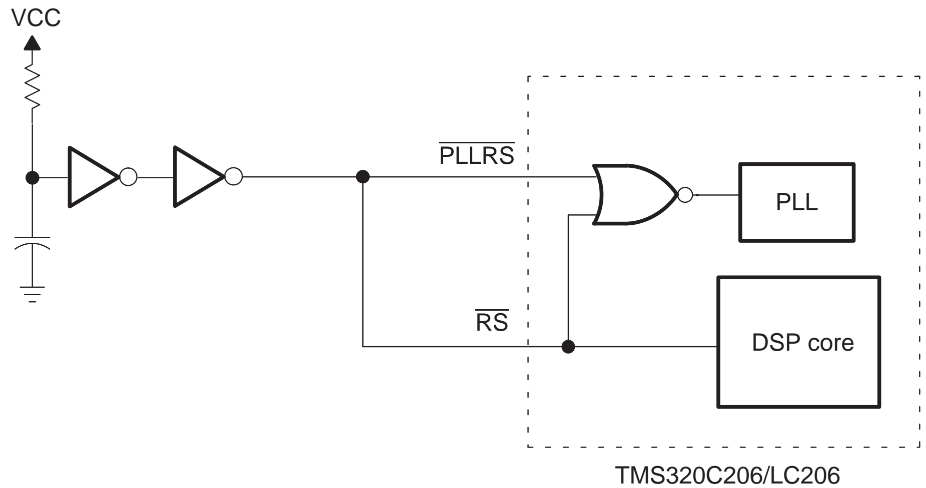
- Case A

The Case A schematic shows initiation of PLL and DSP core reset at power up. After power up, reset pulses on $\overline{\text{RS2}}$ (for example, watchdog timer) reset the DSP core only. The PLL does not reset as $\overline{\text{PLLRS}}$ remains inactive high while $\overline{\text{RS2}}$ is active low. This scheme keeps CLKOUT1 locked for all resets except for power-on reset.



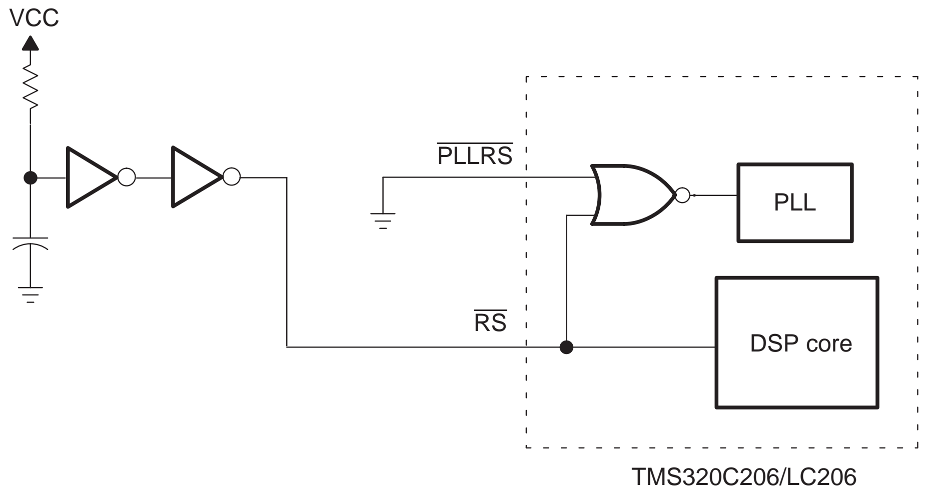
- Case B

The Case B schematic shows initiation of the PLL reset and DSP core reset for every reset. Following every reset, the PLL initiates the PLL locking sequence as $\overline{\text{PLLRS}}$ is low during reset RS.



- Case C

The Case C schematic shown is equivalent to case B. PLL and DSP core are reset for each reset. PLL initiates the locking sequence for every reset as $\overline{\text{PLLRS}}$ is low during reset.



5.8 Power-Down Mode

The ’C20x has a power-down mode that allows the ’C20x core to enter a dormant state and use less power than during normal operation. Executing an IDLE instruction initiates power-down mode. When the IDLE instruction executes, the program counter is incremented once, and then all CPU activities are halted. While the ’C20x is in power-down mode, all of its internal contents are maintained. The content of all on-chip RAM remains unchanged. The peripheral circuits continue to operate, allowing the serial ports and the timer to take the CPU out of the power-down state. The CLKOUT1 pin remains active if bit 0 of the CLK register is set to 0.

The methods for terminating power-down mode depend on whether the power-down was initiated under normal circumstances or as part of a HOLD operation. sections 5.8.1 and 5.8.2 describe the differences.

5.8.1 Normal Termination of Power-Down Mode

If power-down has been initiated, any hardware interrupt (internal or external) takes the processor out of the IDLE state. If you use reset or $\overline{\text{NMI}}$, the CPU will immediately execute the corresponding interrupt service routine. In addition, if you use reset, registers will assume their reset values.

For a maskable hardware interrupt to wake the processor, it must be unmasked by the interrupt mask register (IMR bit = 1). However, if the interrupt is unmasked and is then requested, the processor will leave the IDLE state regardless of the value of the INTM bit (bit 9 of status register ST0). The value of the INTM bit will only determine the action of the CPU *after* power-down has been terminated:

- **INTM = 0.** The interrupt is enabled, and the CPU executes the corresponding interrupt service routine.
- **INTM = 1.** The interrupt is disabled, and the CPU continues with the instruction after IDLE.

If you do not want the CPU to follow an interrupt service routine before continuing with the interrupted program sequence:

- Do not use reset or $\overline{\text{NMI}}$ to bring the processor out of power-down.
- Make sure your program globally disables maskable interrupts (sets INTM to 1) before IDLE is executed.

5.8.2 Termination of Power-Down During a HOLD Operation

One of the necessary steps in the HOLD operation is the execution of an IDLE instruction (see section 4.6, *Direct Memory Access Using The HOLD Operation*, on page 4-18) . There are unique characteristics of the HOLD operation that affect how the IDLE state can be exited.

Before performing a HOLD operation, your program must write a 0 to the MODE bit (bit 4 of the interrupt control register, ICR). This makes the $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ pin both negative- and positive-edge sensitive. A *falling* edge on $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ will cause the CPU to branch to the interrupt service routine, which initiates the HOLD operation with an IDLE instruction. A subsequent *rising* edge on $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ can take the CPU out of the IDLE state and end the HOLD operation. This rising-edge interrupt does *not* cause the CPU to branch to the interrupt service routine.

The recommended software logic for the HOLD operation is described in section 4.6, *Direct Memory Access Using the HOLD Operation.*

During a HOLD operation, there are only three valid methods for taking the CPU out of the IDLE state:

- ❑ Causing a rising edge on the $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ pin.
- ❑ Asserting a system reset at the reset pin.
- ❑ Asserting the nonmaskable interrupt $\overline{\text{NMI}}$ at the $\overline{\text{NMI}}$ pin.

If you use reset or $\overline{\text{NMI}}$, the CPU will immediately execute the corresponding interrupt service routine. In addition, if you use reset, the contents of some registers will be changed. For more information about exiting a HOLD operation with reset or $\overline{\text{NMI}}$, see section 4.6, *Direct Memory Access Using The HOLD Operation.*

Chapter 6

Addressing Modes

This chapter explains the three basic memory addressing modes used by the 'C20x instruction set. The three modes are:

- ❏ Immediate addressing mode
- ❏ Direct addressing mode
- ❏ Indirect addressing mode

In immediate addressing, a constant to be manipulated by the instruction is supplied directly as an operand of that instruction. Two types of immediate addressing are available—short and long. In short-immediate addressing, an 8-, 9-, or 13-bit operand is included in the instruction word. Long-immediate addressing uses a 16-bit operand.

When you need to access data memory, you can use direct or indirect addressing. Direct addressing concatenates seven bits of the instruction word with the nine bits of the data-memory page pointer (DP) to form the 16-bit data memory address. Indirect addressing accesses data memory through one of eight 16-bit auxiliary registers.

Topic **Page**

6.1 Immediate Addressing Mode 6-2

6.2 Direct Addressing Mode 6-4

6.3 Indirect Addressing Mode 6-9

6.1 Immediate Addressing Mode

In immediate addressing, the instruction word contains a constant to be manipulated by the instruction. The 'C20x supports two types of immediate addressing:

- Short-immediate addressing. Instructions that use short-immediate addressing take an 8-bit, 9-bit, or 13-bit constant as an operand. Short-immediate instructions require a single instruction word, with the constant embedded in that word.
- Long-immediate addressing. Instructions that use long-immediate addressing take a 16-bit constant as an operand and require two instruction words. The constant is sent as the second instruction word. This 16-bit value can be used as an absolute constant or as a 2s-complement value.

6.1.1 Examples of Immediate Addressing

In Example 6–1, the immediate operand is contained as a part of the RPT instruction word. For this RPT instruction, the instruction register will be loaded with the value shown in Figure 6–1. Immediate operands are preceded by the symbol #.

Example 6–1. RPT Instruction Using Short-Immediate Addressing

```
RPT #99      ;Execute the instruction that follows RPT
             ;100 times.
```

Figure 6–1. Instruction Register Contents for Example 6–1

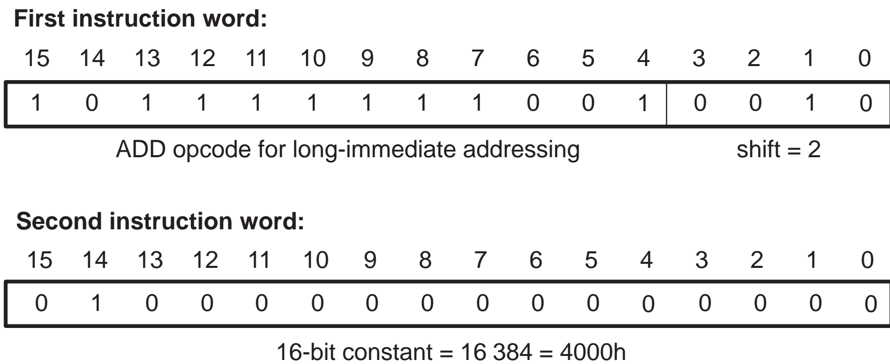
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	0	1	1	0	0	0	1	1
RPT opcode for immediate addressing								8-bit constant = 99							

In Example 6–2, the immediate operand is contained in the second instruction word. The instruction register receives, consecutively, the two 16-bit values shown in Figure 6–2.

Example 6–2. ADD Instruction Using Long-Immediate Addressing

```
ADD    #16384,2  ;Shift the value 16384 left by two bits
                 ;and add the result to the accumulator.
```

Figure 6–2. Two Words Loaded Consecutively to the Instruction Register in Example 6–2



6.2 Direct Addressing Mode

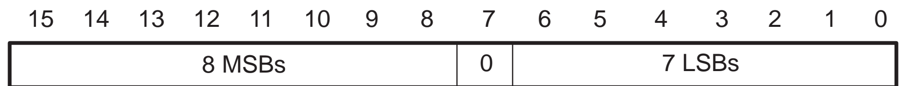
In the direct addressing mode, data memory is addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 data pages labeled 0 through 511, as shown in Figure 6–3. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register ST0. For example, if the DP value is 000000000_2, the current data page is 0. If the DP value is 000000010_2, the current data page is 2.

Figure 6–3. Pages of Data Memory

DP value	Offset	Data Memory
0000 0000 0 : 0000 0000 0	000 0000 : 111 1111	Page 0: 0000h–007Fh
0000 0000 1 : 0000 0000 1	000 0000 : 111 1111	Page 1: 0080h–00FFh
0000 0001 0 : 0000 0001 0	000 0000 : 111 1111	Page 2: 0100h–017Fh
. . .	. . .	. . .
1111 1111 1 : 1111 1111 1	000 0000 : 111 1111	Page 511: FF80h–FFFFh

In addition to the data page, the processor must know the particular word being referenced on that page. This is determined by a 7-bit offset (see Figure 6–3). The offset is supplied by the seven least significant bits (LSBs) of the instruction register, which holds the opcode for the next instruction to be executed. In direct addressing mode, the content of the instruction register has the format shown in Figure 6–4.

Figure 6–4. Instruction Register (IR) Contents in Direct Addressing Mode



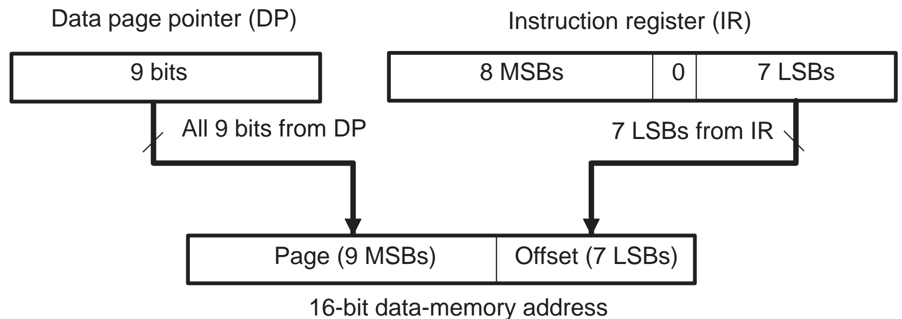
8 MSBs Bits 15 through 8 indicate the instruction type (for example, ADD) and also contain any information regarding a shift of the data value to be accessed by the instruction.

0 **Direct/indirect indicator.** Bit 7 contains a 0 to define the addressing mode as direct.

7 LSBs Bits 6 through 0 indicate the offset for the data-memory address referenced by the instruction.

To form a complete 16-bit address, the processor concatenates the DP value and the seven LSBs of the instruction register, as shown in Figure 6–5. The DP supplies the nine most significant bits (MSBs) of the address (the page number), and the seven LSBs of the instruction register supply the seven LSBs of the address (the offset). For example, to access data address 003Fh, you specify data page 0 (DP = 0000 0000 0) and an offset of 011 1111. Concatenating the DP and the offset produces the 16-bit address 0000 0000 0011 1111, which is 003Fh or decimal 63.

Figure 6–5. Generation of Data Addresses in Direct Addressing Mode



Initialize the DP in All Programs

It is critical that all programs initialize the DP. The DP is not initialized by reset and is undefined after power up. The 'C20x development tools use default values for many parameters, including the DP. However, programs that do not explicitly initialize the DP can execute improperly, depending on whether they are executed on a 'C20x device or with a development tool.

6.2.1 Using Direct Addressing Mode

When you use direct addressing mode, the processor uses the DP to find the data page and uses the seven LSBs of the instruction register to find a particular address on that page. Always do the following:

1) Set the data page. Load the appropriate value (from 0 to 511) into the DP. The DP register can be loaded by the LDP instruction or by any instruction that can load a value to ST0. The LDP instruction loads the DP directly without affecting the other bits of ST0, and it clearly indicates the value loaded into the DP. For example, to set the current data page to 32 (addresses 1000h–107Fh), you can use:

```
LDP #32    ;Initialize data page pointer
```

2) Specify the offset. Supply the 7-bit offset as an operand of the instruction. For example, if you want the ADD instruction to use the value at the second address of the current data page, you would write:

```
ADD 1h     ;Add to accumulator the value in the current
           ;data page, offset of 1.
```

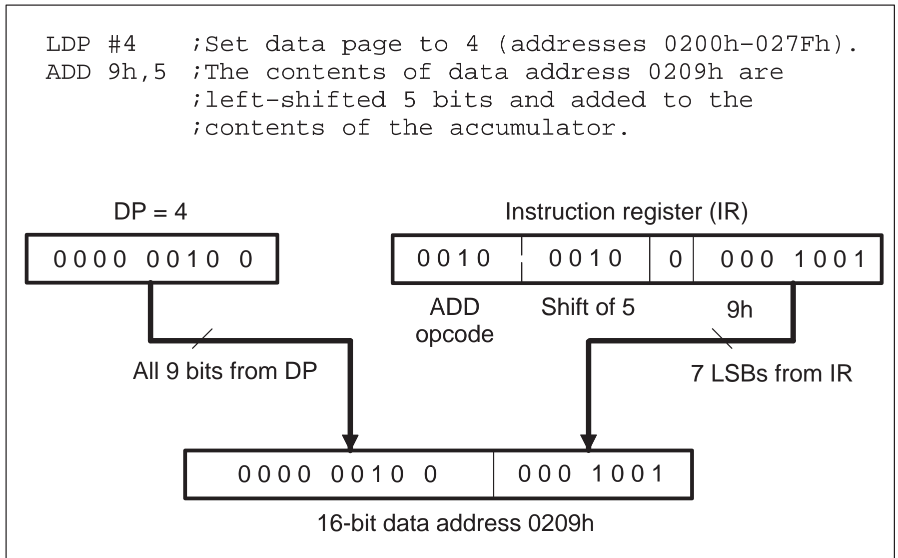
You do not have to set the data page prior to every instruction that uses direct addressing. If all the instructions in a block of code access the same data page, you can simply load the DP at the front of the block. However, if various data pages are being accessed throughout the block of code, be sure the DP is changed whenever a new data page should be accessed.

6.2.2 Examples of Direct Addressing

In Example 6–3, the first instruction loads the DP with 000000100_2 (4) to set the current data page to 4. The ADD instruction then references a data memory address that is generated as shown following the program code. Before the ADD instruction is executed, the opcode is loaded into the instruction register. Together, the DP and the seven LSBs of the instruction register form the complete 16-bit address, 0000001000001001_2 (0209h).

Example 6–3. Using Direct Addressing with ADD (Shift of 0 to 15)

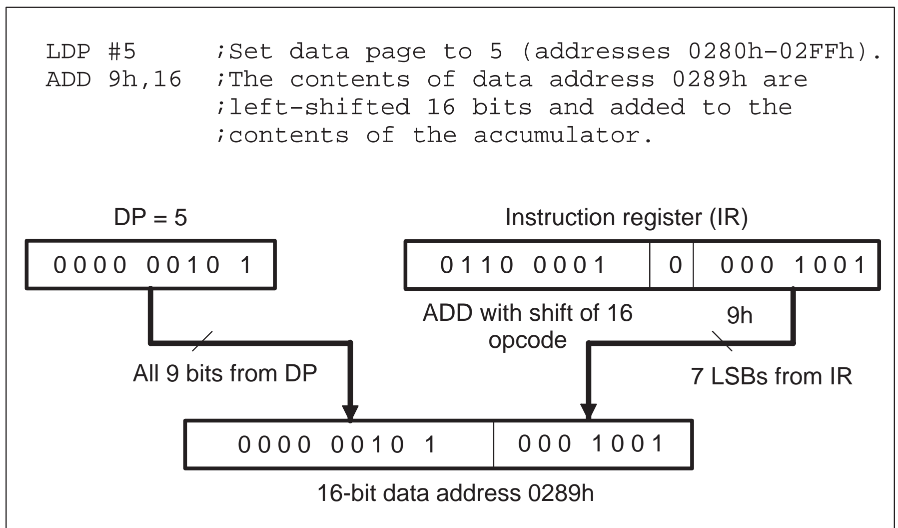
```
LDP #4    ;Set data page to 4 (addresses 0200h–027Fh).
ADD 9h,5  ;The contents of data address 0209h are
          ;left-shifted 5 bits and added to the
          ;contents of the accumulator.
```



In Example 6–4, the ADD instruction references a data memory address that is generated as shown following the program code. For any instruction that performs a shift of 16, the shift value is not embedded directly in the instruction word; instead, all eight MSBs contain an opcode that not only indicates the instruction type but also a shift of 16. The eight MSBs of the instruction word indicate an ADD with a shift of 16.

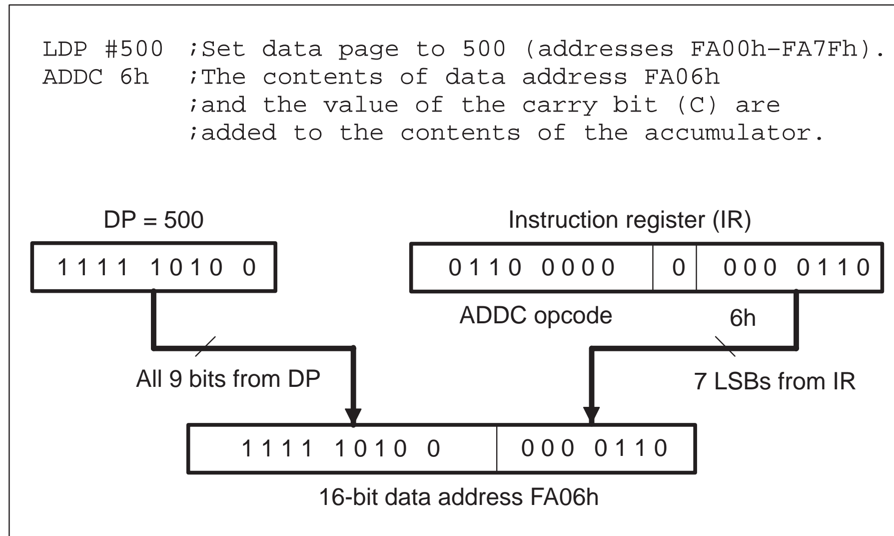
Example 6–4. Using Direct Addressing with ADD (Shift of 16)

```
LDP #5     ;Set data page to 5 (addresses 0280h–02FFh).
ADD 9h,16  ;The contents of data address 0289h are
           ;left-shifted 16 bits and added to the
           ;contents of the accumulator.
```



In Example 6–5, the ADDC instruction references a data memory address that is generated as shown following the program code. Note that if an instruction does not perform shifts, like the ADDC instruction does not, all eight MSBs of the instruction contain the opcode for the instruction type.

Example 6–5. Using Direct Addressing with ADDC



```
LDP #500  ;Set data page to 500 (addresses FA00h–FA7Fh).
ADDC 6h   ;The contents of data address FA06h
          ;and the value of the carry bit (C) are
          ;added to the contents of the accumulator.
```

6.3 Indirect Addressing Mode

Eight auxiliary registers (AR0–AR7) provide flexible and powerful indirect addressing. Any location in the 64K data memory space can be accessed using a 16-bit address contained in an auxiliary register.

6.3.1 Current Auxiliary Register

To select a specific auxiliary register, load the 3-bit auxiliary register pointer (ARP) of status register ST0 with a value from 0 to 7. The ARP can be loaded as a primary operation by the MAR instruction or by the LST instruction. The ARP can be loaded as a secondary operation by any instruction that supports indirect addressing.

The register pointed to by the ARP is referred to as the *current auxiliary register* or *current AR*. During the processing of an instruction, the content of the current auxiliary register is used as the address at which the data-memory access will take place. The ARAU passes this address to the data-read address bus (DRAB) if the instruction requires a read from data memory, or it passes the address to the data-write address bus (DWAB) if the instruction requires a write to data memory. After the instruction uses the data value, the contents of the current auxiliary register can be incremented or decremented by the ARAU, which implements unsigned 16-bit arithmetic.

Normally, the ARAU performs its 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. There is an exception to this rule: During processing of the NORM instruction, the auxiliary register and/or ARP modification is done during the execute phase of the pipeline. For information on the operation of the pipeline, see section 5.2 on page 5-7.

6.3.2 Indirect Addressing Options

The ’C20x provides four types of indirect addressing options:

- No increment or decrement. The instruction uses the content of the current auxiliary register as the data memory address but neither increments nor decrements the content of the current auxiliary register.
- Increment or decrement by 1. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by one.
- Increment or decrement by an index amount. The value in AR0 is the index amount. The instruction uses the content of the current auxiliary register

as the data memory address and then increments or decrements the content of the current auxiliary register by the index amount.

- ❑ Increment or decrement by an index amount using reverse carry. The value in AR0 is the index amount. After the instruction uses the content of the current auxiliary register as the data-memory address, that content is incremented or decremented by the index amount. The addition or subtraction, in this case, is done with the carry propagation reversed (for FFTs).

These four option types provide the seven indirect addressing options listed in Table 6–1. The table also shows the instruction operand that corresponds to each indirect addressing option and gives an example of how each option is used.

Table 6–1. Indirect Addressing Operands

Option	Operand	Example
No increment or decrement	*	**LT *** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR.
Increment by 1	*+	**LT *+** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then adds one to the content of the current AR.
Decrement by 1	*–	**LT *–** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then subtracts one from the content of the current AR.
Increment by index amount	*0+	**LT *0+** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR.
Decrement by index amount	*0–	**LT *0–** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR.

Table 6–1. Indirect Addressing Operands (Continued)

Option	Operand	Example
Increment by index amount, adding with reverse carry	*BR0+	**LT *BR0+** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR, adding with reverse carry propagation.
Decrement by index amount, subtracting with reverse carry	*BR0–	**LT *BR0–** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR, subtracting with bit reverse carry propagation.

All increments or decrements are performed by the auxiliary register arithmetic unit (ARAU) in the same cycle during which the instruction is being decoded in the pipeline.

The bit-reversed indexed addressing 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 when the address is selected, and AR0 is added to or subtracted from the current auxiliary register. A typical use of this addressing mode requires that AR0 first be set to a value corresponding to half of the array's size, and that the current AR value be set to the base address of the data (the first data point).

6.3.3 Next Auxiliary Register

In addition to updating the current auxiliary register, a number of instructions can also specify the *next auxiliary register* or *next AR*. This register will be the current auxiliary register when the instruction execution is complete. The instructions that allow you to specify the next auxiliary register load the ARP with a new value. When the ARP is loaded with that value, the previous ARP value is loaded into the auxiliary register pointer buffer (ARB). Example 6–6 illustrates the selection of a next auxiliary register, as well as other indirect addressing features discussed so far.

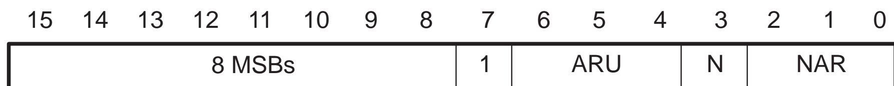
Example 6–6. Selecting a New Current Auxiliary Register

```
MAR*,AR1      ;Load the ARP with 1 to make AR1 the
              ;current auxiliary register.
LT *+,AR2     ;AR2 is the next auxiliary register.
              ;Load the TREG with the content of the
              ;address referenced by AR1, add one to
              ;the content of AR1, then make AR2 the
              ;current auxiliary register.
MPY*          ;Multiply TREG by content of address
              ;referenced by AR2.
```

6.3.4 Indirect Addressing Opcode Format

Figure 6–6 shows the format of the instruction word loaded into the instruction register when you use indirect addressing. The opcode fields are described following the figure.

Figure 6–6. Instruction Register Content in Indirect Addressing



8 MSBs Bits 15 through 8 indicate the instruction type (for example, LT) and also contain any information regarding data shifts.

1 **Direct/indirect indicator.** Bit 7 contains a 1 to define the addressing mode as indirect.

ARU **Auxiliary register update code.** Bits 6 through 4 determine whether and how the current auxiliary register is incremented or decremented. See Table 6–2.

N **Next auxiliary register indicator.** Bit 3 specifies whether the instruction will change the ARP value.

N = 0 If N is 0, the content of the ARP will remain unchanged.

N = 1 If N is 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.

NAR **Next auxiliary register value.** Bits 2 through 0 contain the value of the next auxiliary register. NAR is loaded into the ARP if N = 1.

Table 6–2. Effects of the ARU Code on the Current Auxiliary Register

ARU Code			
6	**5**	**4**	**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 – AR0 → current AR [reverse carry propagation]
1	0	1	current AR – AR0 → current AR
1	1	0	current AR + AR0 → current AR
1	1	1	current AR + AR0 → current AR [reverse carry propagation]

Table 6–3 shows the opcode field bits and the notation used for indirect addressing. It also shows the corresponding operations performed on the current auxiliary register and the ARP.

Table 6–3. Field Bits and Notation for Indirect Addressing

Instruction Opcode Bits												
15	**–**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Operand(s)**	**Operation**
←	8 MSBs	→	1	0	0	0	0	←NAR→			*	No manipulation of current AR
←	8 MSBs	→	1	0	0	0	1	←NAR→			*,AR*n*	NAR → ARP
←	8 MSBs	→	1	0	0	1	0	←NAR→			*–	current AR – 1 → current AR
←	8 MSBs	→	1	0	0	1	1	←NAR→			*–,AR*n*	current AR – 1 → current AR NAR → ARP
←	8 MSBs	→	1	0	1	0	0	←NAR→			*+	current AR + 1 → current AR
←	8 MSBs	→	1	0	1	0	1	←NAR→			*+,AR*n*	current AR + 1 → current AR NAR → ARP
←	8 MSBs	→	1	1	0	0	0	←NAR→			*BR0–	current AR – *rc*AR0 → current AR †
←	8 MSBs	→	1	1	0	0	1	←NAR→			*BR0–,AR*n*	current AR – *rc*AR0 → current AR NAR → ARP †
←	8 MSBs	→	1	1	0	1	0	←NAR→			*0–	current AR – AR0 → current AR
←	8 MSBs	→	1	1	0	1	1	←NAR→			*0–,AR*n*	current AR – AR0 → current AR NAR → ARP
←	8 MSBs	→	1	1	1	0	0	←NAR→			*0+	current AR + AR0 → current AR
←	8 MSBs	→	1	1	1	0	1	←NAR→			*0+,AR*n*	current AR + AR0 → current AR NAR → ARP
←	8 MSBs	→	1	1	1	1	0	←NAR→			*BR0+	current AR + *rc*AR0 → current AR †
←	8 MSBs	→	1	1	1	1	1	←NAR→			*BR0+,AR*n*	current AR + *rc*AR0 → current AR NAR → ARP †

† Bit-reversed addressing mode

Legend:

rc Reverse carry propagation
NAR Next AR
n 0, 1, 2, ..., or 7
8 MSBs Eight bits determined by instruction type and (sometimes) shift information
→ Is loaded into

6.3.5 Examples of Indirect Addressing

In Example 6–7, when the ADD instruction is fetched from program memory, the instruction register is loaded with the value shown.

Example 6–7. No Increment or Decrement

```
ADD *,8         ;Add to the accumulator the content of the
                ;data-memory address referenced by the
                ;current auxiliary register. The data
                ;is left-shifted 8 bits before being added.
```

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

ADD opcode
Shift = 8
Addressing mode = indirect
ARU = No operation on current AR
N = No next AR specified
NAR = don’t cares

In Example 6–8, when the ADD instruction is fetched from program memory, the instruction register is loaded with the value shown.

Example 6–8. Increment by 1

```
ADD *+,8,AR4    ;Operates as in Example 6–7, but
                ;in addition, the current auxiliary
                ;register is incremented by one, and
                ;AR4 is chosen as the next auxiliary
                ;register.
```

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

ADD opcode
Shift = 8
Addressing mode = indirect
ARU = increment current AR by 1
N = next AR specified
NAR = 4

Example 6–9. Decrement by 1

```
ADD *-,8     ;Operates as in Example 6-7, but in
             ;addition, the current auxiliary register
             ;is decremented by one.
```

Example 6–10. Increment by Index Amount

```
ADD *0+,8   ;Operates as in Example 6–7, but in
            ;addition, the content of register AR0
            ;is added to the current auxiliary
            ;register.
```

Example 6–11. Decrement by Index Amount

```
ADD *0–,8    ;Operates as in Example 6–7, but in
             ;addition, the content of register AR0
             ;is subtracted from the current auxiliary
             ;register.
```

Example 6–12. Increment by Index Amount With Reverse Carry Propagation

```
ADD *BR0+,8  ;Operates as in Example 6–10, except that
             ;the content of register AR0 is added to
             ;the current auxiliary register with
             ;reverse carry propagation.
```

Example 6–13. Decrement by Index Amount With Reverse Carry Propagation

```
ADD *BR0–,8  ;Operates as in Example 6–11, except that
             ;the content of register AR0 is subtracted
             ;from the current auxiliary register with
             ;reverse carry propagation.
```

6.3.6 Modifying Auxiliary Register Content

The LAR, ADRK, SBRK, and MAR instructions are specialized instructions for changing the content of an auxiliary register (AR):

- The LAR instruction loads an AR.
- The ADRK instruction adds an immediate value to an AR; SBRK subtracts an immediate value.
- The MAR instruction can increment or decrement an AR value by one or by an index amount.

However, you are not limited to these four instructions. Auxiliary registers can be modified by any instruction that supports indirect addressing operands. (Indirect addressing can be used with all instructions except those that have immediate operands or no operands.)

Chapter 7

Assembly Language Instructions

The ’C20x instruction set supports numerically intensive signal-processing operations as well as general-purpose applications such as multiprocessing and high-speed control. The ’C20x instruction set is compatible with the ’C2x instruction set; code written for the ’C2x can be reassembled to run on the ’C20x. The ’C5x instruction set is a superset of that of the ’C20x; thus, code written for the ’C20x can be upgraded to run on a ’C5x.

This chapter describes the assembly language instructions.

Topic **Page**

7.1 Instruction Set Summary **7-2**

7.2 How To Use the Instruction Descriptions **7-12**

7.3 Instruction Descriptions **7-20**

7.1 Instruction Set Summary

This section provides a summary of the instruction set in six tables (Table 7–1 to Table 7–6) according to the following functional headings:

- ❑ Accumulator, arithmetic, and logic instructions (see Table 7–1 on page 7-4)
- ❑ Auxiliary register and data page pointer instructions (see Table 7–2 on page 7-7)
- ❑ TREG, PREG, and multiply instructions (see Table 7–3 on page 7-8)
- ❑ Branch instructions (see Table 7–4 on page 7-9)
- ❑ Control instructions (see Table 7–5 on page 7-9)
- ❑ I/O and memory operations (see Table 7–6 on page 7-11)

Within each table, the instructions are arranged alphabetically. The number of words that an instruction occupies in program memory is specified in column three of each table; the number of cycles that an instruction requires to execute is in column four. All instructions are assumed to be executed from internal program memory (RAM) and internal data dual-access memory. The cycle timings are for single-instruction execution, not for repeat mode. Additional information about each instruction is presented in the individual instruction descriptions in section 7.2.

For your reference, here are definitions of the symbols used in these six summary tables:

ACC The accumulator

AR Auxiliary register

ARX A 3-bit value used in the LAR and SAR instructions to designate which auxiliary register 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. The CMPR instruction performs a comparison specified by the value of CM:
If CM = 00, test whether current AR = AR0
If CM = 01, test whether current AR < AR0
If CM = 10, test whether current AR > AR0
If CM = 11, test whether current AR ≠ AR0

IAAA AAAA (One I followed by seven As) The I at the left represents a bit that reflects whether direct addressing (I = 0) or indirect addressing (I = 1) is being used. When direct addressing is 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 6.3, *Indirect Addressing Mode*, p. 6-9).

IIII IIII (Eight Is) An 8-bit constant used in short immediate addressing

I IIII IIII (Nine Is) A 9-bit constant used in short immediate addressing for the LDP instruction

I IIII IIII IIII (Thirteen Is) A 13-bit constant used in short immediate addressing for the MPY instruction

I NTR# 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 PM bits of status register ST1 by the SPM instruction

SHF A 3-bit left-shift value

SHFT A 4-bit left-shift value

TP A 2-bit value used by the conditional execution instructions to represent four conditions:

$\overline{\text{BIO}}$ pin low TP = 00
TC bit =1 TP = 01
TC bit = 0 TP = 10
No condition TP = 11

ZLVC ZLVC Two 4-bit fields — each representing the following conditions:

ACC = 0	Z
ACC < 0	L
Overflow	V
Carry	C

A conditional instruction contains two of these 4-bit fields. The 4-LSB field of the instruction is a mask field. A 1 in the corresponding mask bit indicates that condition is being tested. For example, to test for ACC ≥ 0, the Z and L fields are set, and the V and C fields are not set. The Z field is set to test the condition ACC = 0, and the L field is reset to test the condition ACC ≥ 0.The second 4-bit field (bits 4 – 7) indicates the state of the conditions to test. The conditions possible with these eight bits are shown in the descriptions for the BCND, CC, and RETC instructions.

+ 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.

Table 7–1. Accumulator, Arithmetic, and Logic Instructions

Mnemonic	Description	Words	Cycles	Opcode
ABS	Absolute value of ACC	1	1	1011 1110 0000 0000
ADD	Add to ACC with shift of 0 to 15, direct or indirect	1	1	0010 SHFT IAAA AAAA
	Add to ACC with shift 0 to 15, long immediate	2	2	1011 1111 1001 SHFT + 1 word
	Add to ACC with shift of 16, direct or indirect	1	1	0110 0001 IAAA AAAA
	Add to ACC, short immediate	1	1	1011 1000 IIII IIII
ADDC	Add to ACC with carry, direct or indirect	1	1	0110 0000 IAAA AAAA
ADDS	Add to low ACC with sign-extension suppressed, direct or indirect	1	1	0110 0010 IAAA AAAA
ADDT	Add to ACC with shift (0 to 15) specified by TREG, direct or indirect	1	1	0110 0011 IAAA AAAA

Table 7–1. Accumulator, Arithmetic, and Logic Instructions (Continued)

Mnemonic	Description	Words	Cycles	Opcode
AND	AND ACC with data value, direct or indirect	1	1	0110 1110 IAAA AAAA
	AND with ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1011 SHFT + 1 word
	AND with ACC with shift of 16, long immediate	2	2	1011 1110 1000 0001 + 1 word
CMPL	Complement ACC	1	1	1011 1110 0000 0001
LACC	Load ACC with shift of 0 to 15, direct or indirect	1	1	0001 SHFT IAAA AAAA
	Load ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1000 SHFT + 1 word
	Load ACC with shift of 16, direct or indirect	1	1	0110 1010 IAAA AAAA
LACL	Load low word of ACC, direct or indirect	1	1	0110 1001 IAAA AAAA
	Load low word of ACC, short immediate	1	1	1011 1001 IIII IIII
LACT	Load ACC with shift (0 to 15) specified by TREG, direct or indirect	1	1	0110 1011 IAAA AAAA
NEG	Negate ACC	1	1	1011 1110 0000 0010
NORM	Normalize the contents of ACC, indirect	1	1	1010 0000 IAAA AAAA
OR	OR ACC with data value, direct or indirect	1	1	0110 1101 IAAA AAAA
	OR with ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1100 SHFT + 1 word
	OR with ACC with shift of 16, long immediate	2	2	1011 1110 1000 0010 + 1 word
ROL	Rotate ACC left	1	1	1011 1110 0000 1100
ROR	Rotate ACC right	1	1	1011 1110 0000 1101
SACH	Store high ACC with shift of 0 to 7, direct or indirect	1	1	1001 1SHF IAAA AAAA
SACL	Store low ACC with shift of 0 to 7, direct or indirect	1	1	1001 0SHF IAAA AAAA
SFL	Shift ACC left	1	1	1011 1110 0000 1001
SFR	Shift ACC right	1	1	1011 1110 0000 1010

Table 7–1. Accumulator, Arithmetic, and Logic Instructions (Continued)

Mnemonic	Description	Words	Cycles	Opcode
SUB	Subtract from ACC with shift of 0 to 15, direct or indirect	1	1	0011 SHFT IAAA AAAA
	Subtract from ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1010 SHFT + 1 word
	Subtract from ACC with shift of 16, direct or indirect	1	1	0110 0101 IAAA AAAA
	Subtract from ACC, short immediate	1	1	1011 1010 IIII IIII
SUBB	Subtract from ACC with borrow, direct or indirect	1	1	0110 0100 IAAA AAAA
SUBC	Conditional subtract, direct or indirect	1	1	0000 1010 IAAA AAAA
SUBS	Subtract from ACC with sign-extension suppressed, direct or indirect	1	1	0110 0110 IAAA AAAA
SUBT	Subtract from ACC with shift (0 to 15) specified by TREG, direct or indirect	1	1	0110 0111 IAAA AAAA
XOR	Exclusive OR ACC with data value, direct or indirect	1	1	0110 1100 IAAA AAAA
	Exclusive OR with ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1101 SHFT + 1 word
	Exclusive OR with ACC with shift of 16, long immediate	2	2	1011 1110 1000 0011 + 1 word
ZALR	Zero low ACC and load high ACC with rounding, direct or indirect	1	1	0110 1000 IAAA AAAA

Table 7–2. Auxiliary Register Instructions

Mnemonic	Description	Words	Cycles	Opcode
ADRK	Add constant to current AR, short immediate	1	1	0111 1000 IIII IIII
BANZ	Branch on current AR not-zero, indirect	2	4 (condition true) 2 (condition false)	0111 1011 1AAA AAAA + 1 word
CMPR	Compare current AR with AR0	1	1	1011 1111 0100 01CM
LAR	Load specified AR from specified data location, direct or indirect	1	2	0000 0ARX IAAA AAAA
	Load specified AR with constant, short immediate	1	2	1011 0ARX IIII IIII
	Load specified AR with constant, long immediate	2	2	1011 1111 0000 1ARX + 1 word
MAR	Modify current AR and/or ARP, indirect (performs no operation when direct)	1	1	1000 1011 IAAA AAAA
SAR	Store specified AR to specified data location, direct or indirect	1	1	1000 0ARX IAAA AAAA
SBRK	Subtract constant from current AR, short immediate	1	1	0111 1100 IIII IIII

Table 7–3. TREG, PREG, and Multiply Instructions

Mnemonic	Description	Words	Cycles	Opcode
APAC	Add PREG to ACC	1	1	1011 1110 0000 0100
LPH	Load high PREG, direct or indirect	1	1	0111 0101 IAAA AAAA
LT	Load TREG, direct or indirect	1	1	0111 0011 IAAA AAAA
LTA	Load TREG and accumulate previous product, direct or indirect	1	1	0111 0000 IAAA AAAA
LTD	Load TREG, accumulate previous product, and move data, direct or indirect	1	1	0111 0010 IAAA AAAA
LTP	Load TREG and store PREG in accumulator, direct or indirect	1	1	0111 0001 IAAA AAAA
LTS	Load TREG and subtract previous product, direct or indirect	1	1	0111 0100 IAAA AAAA
MAC	Multiply and accumulate, direct or indirect	2	3	1010 0010 IAAA AAAA + 1 word
MACD	Multiply and accumulate with data move, direct or indirect	2	3	1010 0011 IAAA AAAA + 1 word
MPY	Multiply TREG by data value, direct or indirect	1	1	0101 0100 IAAA AAAA
	Multiply TREG by 13-bit constant, short immediate	1	1	110I IIII IIII IIII
MPYA	Multiply and accumulate previous product, direct or indirect	1	1	0101 0000 IAAA AAAA
MPYS	Multiply and subtract previous product, direct or indirect	1	1	0101 0001 IAAA AAAA
MPYU	Multiply unsigned, direct or indirect	1	1	0101 0101 IAAA AAAA
PAC	Load ACC with PREG	1	1	1011 1110 0000 0011
SPAC	Subtract PREG from ACC	1	1	1011 1110 0000 0101
SPH	Store high PREG, direct or indirect	1	1	1000 1101 IAAA AAAA
SPL	Store low PREG, direct or indirect	1	1	1000 1100 IAAA AAAA
SPM	Set product shift mode	1	1	1011 1111 0000 00PM
SQRA	Square and accumulate previous product, direct or indirect	1	1	0101 0010 IAAA AAAA
SQRS	Square and subtract previous product, direct or indirect	1	1	0101 0011 IAAA AAAA

Table 7–4. Branch Instructions

Mnemonic	Description	Words	Cycles	Opcode
B	Branch unconditionally, indirect	2	4	0111 1001 1AAA AAAA + 1 word
BACC	Branch to address specified by ACC	1	4	1011 1110 0010 0000
BANZ	Branch on current AR not-zero, indirect	2	4 (condition true) 2 (condition false)	0111 1011 1AAA AAAA + 1 word
BCND	Branch conditionally	2	4 (conditions true) 2 (any condition false)	1110 00TP ZLVC ZLVC + 1 word
CALA	Call subroutine at location specified by ACC	1	4	1011 1110 0011 0000
CALL	Call subroutine, indirect	2	4	0111 1010 1AAA AAAA + 1 word
CC	Call conditionally	2	4 (conditions true) 2 (any condition false)	1110 10TP ZLVC ZLVC + 1 word
INTR	Soft interrupt	1	4	1011 1110 011I NTR#
NMI	Nonmaskable interrupt	1	4	1011 1110 0101 0010
RET	Return from subroutine	1	4	1110 1111 0000 0000
RETC	Return conditionally	1	4 (conditions true) 2 (any condition false)	1110 11TP ZLVC ZLVC
TRAP	Software interrupt	1	4	1011 1110 0101 0001

Table 7–5. Control Instructions

Mnemonic	Description	Words	Cycles	Opcode
BIT	Test bit, direct or indirect	1	1	0100 BITX IAAA AAAA
BITT	Test bit specified by TREG, direct or indirect	1	1	0110 1111 IAAA AAAA
CLRC	Clear C bit	1	1	1011 1110 0100 1110
	Clear CNF bit	1	1	1011 1110 0100 0100
	Clear INTM bit	1	1	1011 1110 0100 0000
	Clear OVM bit	1	1	1011 1110 0100 0010
	Clear SXM bit	1	1	1011 1110 0100 0110
	Clear TC bit	1	1	1011 1110 0100 1010
	Clear XF bit	1	1	1011 1110 0100 1100

Table 7–5. Control Instructions (Continued)

Mnemonic	Description	Words	Cycles	Opcode
IDLE	Idle until interrupt	1	1	1011 1110 0010 0010
LDP	Load data page pointer, direct or indirect	1	2	0000 1101 IAAA AAAA
	Load data page pointer, short immediate	1	2	1011 110I IIII IIII
LST	Load status register ST0, direct or indirect	1	2	0000 1110 IAAA AAAA
	Load status register ST1, direct or indirect	1	2	0000 1111 IAAA AAAA
NOP	No operation	1	1	1000 1011 0000 0000
POP	Pop top of stack to low ACC	1	1	1011 1110 0011 0010
POPD	Pop top of stack to data memory, direct or indirect	1	1	1000 1010 IAAA AAAA
PSHD	Push data memory value on stack, direct or indirect	1	1	0111 0110 IAAA AAAA
PUSH	Push low ACC onto stack	1	1	1011 1110 0011 1100
RPT	Repeat next instruction, direct or indirect	1	1	0000 1011 IAAA AAAA
	Repeat next instruction, short immediate	1	1	1011 1011 IIII IIII
SETC	Set C bit	1	1	1011 1110 0100 1111
	Set CNF bit	1	1	1011 1110 0100 0101
	Set INTM bit	1	1	1011 1110 0100 0001
	Set OVM bit	1	1	1011 1110 0100 0011
	Set SXM bit	1	1	1011 1110 0100 0111
	Set TC bit	1	1	1011 1110 0100 1011
	Set XF bit	1	1	1011 1110 0100 1101
SPM	Set product shift mode	1	1	1011 1111 0000 00PM
SST	Store status register ST0, direct or indirect	1	1	1000 1110 IAAA AAAA
	Store status register ST1, direct or indirect	1	1	1000 1111 IAAA AAAA

Table 7–6. I/O and Memory Instructions

Mnemonic	Description	Words	Cycles	Opcode
BLDD	Block move from data memory to data memory, direct/indirect with long immediate source	2	3	1010 1000 IAAA AAAA + 1 word
	Block move from data memory to data memory, direct/indirect with long immediate destination	2	3	1010 1001 IAAA AAAA + 1 word
BLPD	Block move from program memory to data memory, direct/indirect with long immediate source	2	3	1010 0101 IAAA AAAA + 1 word
DMOV	Data move in data memory, direct or indirect	1	1	0111 0111 IAAA AAAA
IN	Input data from I/O location, direct or indirect	2	2	1010 1111 IAAA AAAA + 1 word
OUT	Output data to port, direct or indirect	2	3	0000 1100 IAAA AAAA + 1 word
SPLK	Store long immediate to data memory location, direct or indirect	2	2	1010 1110 IAAA AAAA + 1 word
TBLR	Table read, direct or indirect	1	3	1010 0110 IAAA AAAA
TBLW	Table write, direct or indirect	1	3	1010 0111 IAAA AAAA

7.2 How To Use the Instruction Descriptions

Section 7.3 contains detailed information on the instruction set. The description for each instruction presents the following categories of information:

- Syntax
- Operands
- Opcode
- Execution
- Status Bits
- Description
- Words
- Cycles
- Examples

7.2.1 Syntax

Each instruction begins with a list of the available assembler syntax expressions and the addressing mode type(s) for each expression. For example, the description for the ADD instruction begins with:

ADD *dma* [, *shift*]	Direct addressing
ADD *dma*, **16**	Direct with left shift of 16
ADD *ind* [, *shift* [, **AR***n*]]	Indirect addressing
ADD *ind*, **16** [, **AR***n*]	Indirect with left shift of 16
ADD #*k*	Short immediate addressing
ADD #*lk* [, *shift*]	Long immediate addressing

These are the notations used in the syntax expressions:

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
```

boldface characters — Boldface characters in an instruction syntax must 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
```

[, x] Operand x is optional.

Example: For the syntax:

ADD *dma*, [, *shift*]

you must supply *dma*, as in the instruction:

`ADD 7h`

and 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`

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. But `RPT 15` uses direct addressing. The number of times the next instruction repeats is determined by a value stored in memory.

Finally, consider this code example:

```
MoveData  BLDD  DAT5, #310h  ;move data at address
                             ;referenced by DAT5 to address
                             ;310h.
```

Note the optional label `MoveData` used as a reference in front of the instruction mnemonic. Place labels either before the instruction mnemonic on the same line or on the preceding line in the first column. (Be sure there are no spaces in your labels.) An optional comment field can conclude the syntax expression. At least one space is required between fields (label, mnemonic, operand, and comment).

7.2.2 Operands

Operands can be constants, or assembly-time expressions referring to memory, I/O ports, register addresses, pointers, shift counts, and a variety of other constants. The operands category for each instruction description defines the variables used for and/or within operands in the syntax expressions. For example, for the ADD instruction, the syntax category gives these syntax expressions:

ADD *dma* [, *shift*]	Direct addressing
ADD *dma*, **16**	Direct with left shift of 16
ADD *ind* [, *shift* [, **AR***n*]]	Indirect addressing
ADD *ind*, **16** [, **AR***n*]	Indirect with left shift of 16
ADD #*k*	Short immediate addressing
ADD #*lk* [, *shift*]	Long immediate addressing

The operands category defines the variables *dma*, *shift*, *ind*, *n*, *k*, and *lk*. For *ind*, an indirect addressing variable, you supply one of the following seven symbols:

*   *+   *–   *0+   *0–   *BR0+   *BR0–

These symbols are defined in section 6.3.2, *Indirect Addressing Options*, on page 6-9.

7.2.3 Opcode

The opcode category breaks down the various bit fields that make up each instruction word. When one of the fields contains a constant value derived directly from an operand, it has the same name as that operand. The contents of fields that do not directly relate to operands have other names; the opcode category either explains these names directly or refers you to a section of this book that explains them in detail. For example, these opcodes are given for the ADDC instruction:

ADDC *dma*

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						

ADDC *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in Section 6.3, *Indirect Addressing Mode* (page 6-9).

The field called dma contains the value *dma*, which is defined in the operands category. The contents of the fields ARU, N, and NAR are derived from the operands *ind* and *n* but do not directly correspond to those operands; therefore, a note directs you to the appropriate section for more details.

7.2.4 Execution

The execution category presents an instruction operation sequence that describes the processing that takes place when the instruction is executed. If the execution event or events depend on the addressing mode used, the execution category specifies which events are associated with which addressing modes. Here are notations used in the execution category:

(r) The content of register or location r.
Example: (ACC) represents the value in the accumulator.

x → y Value x is assigned to register or location y.
Example: (data-memory address) → ACC means: The content of the specified data-memory address is put into the accumulator.

r(n:m) Bits n through m of register or location r.
Example: ACC(15:0) represents 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)) represents the content of bits 31 through 16 of the accumulator.

nnh Indicates that nn represents a hexadecimal number.

7.2.5 Status Bits

The bits in status registers ST0 and ST1 affect the operation of certain instructions and are affected by certain instructions. The status bits category of each instruction description states which of the bits (if any) affect the execution of the instruction and which of the bits (if any) are affected by the instruction.

7.2.6 Description

The description category explains what happens during instruction execution and its effect on the rest of the processor or on memory contents. It also discusses any constraints on the operands imposed by the processor or the assembler. This description parallels and supplements the information given in the execution category.

7.2.7 Words

The words category specifies the number of memory words (one or two) required to store the instruction. When the number of words depends on the addressing mode used for an instruction, the words category specifies which addressing modes require one word and which require two words.

7.2.8 Cycles

The cycles category of each instruction description contains tables showing the number of processor machine cycles (CLKOUT1 periods) required for the instruction to execute in a given memory configuration when executed as a single instruction or when repeated with the RPT instruction. For example:

Cycles for a Single Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 these tables indicate the program source location, defined as follows:

ROM The instruction executes from internal program ROM.

DARAM The instruction executes from internal dual-access program RAM.

SARAM The instruction executes from internal single-access program RAM.

External The instruction executes from external program memory.

If an instruction requires memory operand(s), the rows in the table indicate the location(s) of the operand(s), as defined here:

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.

For the RPT mode execution, *n* indicates the number of times a given instruction is repeated by an RPT instruction. Additional cycles (wait states) can be generated for program-memory, data-memory, and I/O accesses by the wait-state generator or by the external READY signal. These additional wait states are represented in the tables by the following variables:

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

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

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

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

If there are multiple accesses to one of the spaces, the variable will be preceded by the appropriate integer multiple. For example, two accesses to external program memory would require 2p wait states. The above variables may also use the subscripts *src, dst,* and *code* to indicate source, destination, and code, respectively.

Single access RAM (SARAM) allows for only one access per cycle. However, the internal single access memory on each ’C20x processor is divided into 2K-word blocks contiguous in address space. You can use SARAM for simultaneous accesses to program memory and data memory if the accesses are made to different 2K-word blocks.

All external reads take at least one machine cycle while all external writes take 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. If the wait state generator or the READY pin is used to add *m* ($m > 0$) wait states to an external access, then external reads require m+1 cycles, and external write accesses require m+2 cycles. See Section 8.5, *Wait-State Generator*, page 8-15, for the discussion on generating wait states.

The instruction-cycle timings are based on the following assumptions:

- At least the next four instructions are fetched from the same memory section (internal or external) that was used to fetch the current instruction (except in the case of PC discontinuity instructions, such as B, CALL, etc.)
- In the single-execution mode, 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 Section 5.2, *Pipeline*, on page 5-7 for more information about pipeline operation.
- In the repeat execution mode, all conflicts caused by the pipelined execution of an instruction are considered.

7.2.9 Examples

Example code is included for each instruction. The effect of the code on memory and/or registers is summarized. Program code is shown in a `special typeface`. The sample code is then followed by a verbal or graphic description of the effect of that code. Consider this example of the ADD instruction:

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

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

Here are the facts and events represented in this example:

- The auxiliary register pointer (ARP) points to the current auxiliary register. Because ARP = 4, the current auxiliary register is AR4.
- When the addition takes place, the CPU follows AR4 to data-memory address 0302h. The content of that address, 2h, is added to the content of the accumulator, also 2h. The result (4h) is placed in the accumulator. (Because the second operand of the instruction specifies a left shift of 0, the data-memory value is not shifted before being added to the accumulator value.)
- The instruction specifies an increment of one for the contents of the current auxiliary register (*+); therefore, after the addition is performed, the content of AR4 is incremented to 0303h.

- The instruction also specifies that AR0 will be the next auxiliary register; therefore, after the instruction ARP = 0.
- Because no carry is generated during the addition, the carry bit (C) becomes 0.

7.3 Instruction Descriptions

This section contains detailed information on the instruction set for the ’C20x (For a summary of the instruction set, see Section 7.1.) The instructions are presented alphabetically, and the description for each instruction presents the following categories of information:

- Syntax
- Operands
- Opcode
- Execution
- Status Bits
- Description
- Words
- Cycles
- Examples

For a description of how to use each of these categories, see Section 7.2.

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 Increment PC, then ...
|(ACC)| → ACC; 0 → C

Status Bits

Affected by	*Affects*
OVM	C and OV

This instruction is not affected by SXM

Description If the contents of the accumulator are greater than or equal to zero, the accumulator is unchanged by the execution of ABS. If the contents of the accumulator are less than zero, the accumulator is replaced by its 2s-complement value. The carry bit (C) on the ’C20x is always reset to zero by the execution of this instruction.

Note that 8000 0000h is a special case. When the overflow mode is not set (OVM = 0), the ABS of 8000 0000h is 8000 0000h. When the overflow mode is set (OVM = 1), the ABS of 8000 0000h is 7FFF FFFFh. In either case, the OV status bit is set.

Words 1

Cycles

Cycles for a Single ABS Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an ABS Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example 1

```
ABS
```

	Before Instruction			After Instruction	
ACC	C: X	1234h	ACC	C: 0	1234h

Example 2

```
ABS
```

	Before Instruction			After Instruction	
ACC	C: X	0FFFFFFFFh	ACC	C: 0	1h

Example 3

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

	Before Instruction			After Instruction	
ACC	C: X	80000000h	ACC	C: 0	7FFFFFFFh
	OV: X			OV: 1	

Example 4

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

	Before Instruction			After Instruction	
ACC	C: X	80000000h	ACC	C: 0	80000000h
	OV: X			OV: 1	

Syntax

ADD *dma* [, *shift*]	Direct addressing
ADD *dma*, **16**	Direct with left shift of 16
ADD *ind* [, *shift* [, **AR***n*]]	Indirect addressing
ADD *ind*, **16** [, **AR***n*]	Indirect with left shift of 16
ADD #*k*	Short immediate addressing
ADD #*lk* [, *shift*]	Long immediate addressing

Operands

dma:	7 LSBs of the data-memory address
shift:	Left shift value from 0 to 15 (defaults to 0)
n:	Value from 0 to 7 designating the next auxiliary register
k:	8-bit short immediate value
lk:	16-bit long immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

ADD *dma* [, *shift*]

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

ADD *dma*, **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						

ADD *ind* [, *shift* [, **AR***n*]]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

ADD *ind*, **16** [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

ADD #*k*

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

ADD #*lk* [, *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	shift			
lk															

Execution

Increment PC, then ...

Event	*Addressing mode*
(ACC) + ((data-memory address) $\times$ 2^{shift}) $\rightarrow$ ACC	Direct or indirect
(ACC) + ((data-memory address) $\times$ 2^{16}) $\rightarrow$ ACC	Direct or indirect (shift of 16)
(ACC) + k $\rightarrow$ ACC	Short immediate
(ACC) + lk $\times$ 2^{shift} $\rightarrow$ ACC	Long immediate

Status Bits

Affected by	*Affects*	*Addressing mode*
SXM and OVM	C and OV	Direct or indirect
OVM	C and OV	Short immediate
SXM and OVM	C and OV	Long immediate

Description

The content of the addressed data memory location or an immediate constant is left-shifted and added to the accumulator. During shifting, low-order bits are zero filled. High-order bits are sign extended if SXM = 1 and zero filled if SXM = 0. The result is stored in the accumulator. When short immediate addressing is used, the addition is unaffected by SXM and is not repeatable.

If you are using indirect addressing and update the ARP, you must specify a shift operand. However, if you do not want a shift to occur, enter a 0 for this operand. For example:

```
ADD *+,0,AR2
```

Normally, the carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry. However, when adding with a shift of 16, the carry bit is set if a carry is generated but otherwise, the carry bit is unaffected. This allows the accumulator to generate the proper single carry when adding a 32-bit number to the accumulator.

Words

Words	*Addressing mode*
1	Direct, indirect, or short immediate
2	Long immediate

Cycles

Cycles for a Single ADD Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an ADD Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 ADD Instruction (Using Short Immediate Addressing)

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Single ADD Instruction (Using Long Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+2p

Example 1

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

	Before Instruction		After Instruction
Data Memory 301h	1h	Data Memory 301h	1h
ACC	C: X, 2h	ACC	C: 0, 04h

Example 2

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

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

Example 3

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

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

Example 4

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

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

Syntax

ADDC *dma* Direct addressing
ADDC *ind* [, **AR***n*] Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

ADDC *dma*

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						

ADDC *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) + (data-memory address) + (C) → ACC

Status Bits

Affected by	Affects
OVM	C and OV

This instruction is not affected by SXM.

Description

The contents of the addressed data-memory location and the value of the carry bit are added to the accumulator with sign extension suppressed. The carry bit is then affected in the normal manner: the carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.

The ADDC instruction can be used in performing multiple-precision arithmetic.

Words

1

Cycles

Cycles for a Single ADDC Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an ADDC Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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     DAT300    ;(DP = 6: addresses 0300h–037Fh;
                   ;DAT300 is a label for 300h)
```

	Before Instruction		After Instruction
Data Memory 300h	04h	Data Memory 300h	04h
ACC	C: 1, 13h	ACC	C: 0, 18h

Example 2

```
ADDC     *–,AR4    ;(OVM = 0)
```

	Before Instruction		After Instruction
ARP	0	ARP	4
AR0	300h	AR0	299h
Data Memory 300h	0h	Data Memory 300h	0h
ACC	C: 1, 0FFFFFFFFh	ACC	C: 1, 0h
OV	X	OV	0

Syntax

ADDS *dma*	Direct addressing
ADDS *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

ADDS *dma*

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						

ADDS *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) + (data-memory address) → ACC

Status Bits

Affected by	*Affects*
OVM	C and OV

This instruction is not affected by SXM.

Description

The contents of the specified data-memory location are added to the accumulator with sign extension suppressed. The data is treated as an unsigned 16-bit number, regardless of SXM. The accumulator contents are treated as a signed number. Note that ADDS produces the same results as an ADD instruction with SXM = 0 and a shift count of 0.

The carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.

Words

1

Cycles

Cycles for a Single ADDS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an ADDS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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    0          ;(DP = 6: addresses 0300h–037Fh)
```

	Before Instruction		After Instruction
Data Memory 300h	0F006h	Data Memory 300h	0F006h
ACC	C: X, 00000003h	ACC	C: 0, 0000F009h

Example 2

```
ADDS    *
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	0300h	AR0	0300h
Data Memory 300h	0FFFFh	Data Memory 300h	0FFFFh
ACC	C: X, 7FFF0000h	ACC	C: 0, 7FFFFFFFh

Syntax

ADDT *dma* — Direct addressing
ADDT *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

ADDT *dma*

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						

ADDT *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) + [(data-memory address) $\times$ $2^{(TREG(3:0))}$] $\rightarrow$ (ACC)

Status Bits

Affected by	Affects
SXM and OVM	C and OV

Description

The data-memory value is left shifted and added to the accumulator, and the result replaces the accumulator contents. The left shift is defined by the four LSBs of the TREG, resulting in shift options from 0 to 15 bits. Sign extension on the data-memory value is controlled by SXM. The carry bit (C) is set when a carry is generated out of the MSB of the accumulator; if no carry is generated, the carry bit is cleared.

Words

1

Cycles

Cycles for a Single ADDT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an ADDT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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      127       ;(DP = 4: addresses 0200h–027Fh,
                    ;SXM = 0)
```

		Before Instruction			After Instruction
Data Memory 027Fh		09h	Data Memory 027Fh		09h
TREG		0FF94h	TREG		0FF94h
ACC	X (C)	0F715h	ACC	0 (C)	0F7A5h

Example 2

```
ADDT      *–,AR4    ;(SXM = 0)
```

		Before Instruction			After Instruction
ARP		0	ARP		4
AR0		027Fh	AR0		027Eh
Data Memory 027Fh		09h	Data Memory 027Fh		09h
TREG		0FF94h	TREG		0FF94h
ACC	X (C)	0F715h	ACC	0 (C)	0F7A5h

Syntax **ADRK #*k*** Short immediate addressing

Operands k: 8-bit short immediate value

Opcode

ADRK #*k*

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

Execution

Increment PC, then ...
(current AR) + 8-bit positive constant → current AR

Status Bits None

Description

The 8-bit immediate value is added, right justified, to the current auxiliary register (the one specified by the current ARP value) and the result replaces the auxiliary register contents. The addition takes place in the ARAU, with the immediate value treated as an 8-bit positive integer. All arithmetic operations on the auxiliary registers are unsigned.

Words 1

Cycles

Cycles for a Single ADRK Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Example

```
ADRK    #80h
```

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

Syntax

AND *dma*	Direct addressing
AND *ind* [, **AR***n*]	Indirect addressing
AND #*lk* [, *shift*]	Long immediate addressing
AND #*lk*, **16**	Long immediate with left shift of 16

Operands

dma:	7 LSBs of the data-memory address
shift:	Left shift value from 0 to 15 (defaults to 0)
n:	Value from 0 to 7 designating the next auxiliary register
lk:	16-bit long immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

AND *dma*

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

AND *ind* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

AND #*lk* [, *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	1	shift			
lk															

AND #*lk*, **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	0	1
lk															

Execution

Increment PC, then ...

Event(s)	*Addressing mode*
(ACC(15:0)) AND (data-memory address) → ACC(15:0) 0 → ACC(31:16)	Direct or indirect
(ACC(31:0)) AND lk $\times$ 2^{shift} → ACC	Long immediate
(ACC(31:0)) AND lk $\times$ 2^{16} → ACC	Long immediate with left shift of 16

Status Bits None

This instruction is not affected by SXM.

Description If direct or indirect addressing is used, the low word of the accumulator is ANDed with a data-memory value, and the result is placed in the low word position in the accumulator. The high word of the accumulator is zeroed. If immediate addressing is used, the long-immediate constant can be shifted. During the shift, low-order and high-order bits not filled by the shifted value are zeroed. The resulting value is ANDed with the accumulator contents.

Words

Words	*Addressing mode*
1	Direct or indirect
2	Long immediate

Cycles

Cycles for a Single AND Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an AND Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 AND Instruction (Using Long Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+2p

Example 1

```
AND     16      ;(DP = 4: addresses 0200h–027Fh)
```

	Before Instruction		After Instruction
Data Memory 0210h	00FFh	Data Memory 0210h	00FFh
ACC	12345678h	ACC	00000078h

Example 2

```
AND     *
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	0301h	AR0	0301h
Data Memory 0301h	0FF00h	Data Memory 0301h	0FF00h
ACC	12345678h	ACC	00005600h

Example 3

```
AND     #00FFh,4
```

	Before Instruction		After Instruction
ACC	12345678h	ACC	00000670h

Syntax **APAC**

Operands None

Opcode

APAC

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 Increment PC, then ...
(ACC) + shifted (PREG) → ACC

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

This instruction is not affected by SXM.

Description The contents of PREG are shifted as defined by the PM status bits of the ST1 register (see Table 7–7) and added to the contents of the accumulator. The result is placed in the accumulator. APAC is not affected by the SXM bit of the status register. PREG is always sign extended. The task of the APAC instruction is also performed as a subtask of the LTA, LTD, MAC, MACD, MPYA, and SQRA instructions.

Table 7–7. Product Shift Modes

PM Bits		
Bit 1	**Bit 0**	**Resulting Shift**
0	0	No shift
0	1	Left shift of 1 bit
1	0	Left shift of 4 bits
1	1	Right shift of 6 bits

Words 1

Cycles

Cycles for a Single APAC Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an APAC Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
APAC        ;(PM = 01)
```

		Before Instruction			After Instruction
PREG		40h	PREG		40h
ACC	X	20h	ACC	0	A0h
	C			C	

Syntax **B** *pma* [, *ind* [, **AR***n*]] Indirect addressing

Operands

pma: 16-bit program-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

B *pma* [, *ind* [, **AR***n*]]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

pma → PC
Modify (current AR) and (ARP) as specified.

Status Bits None

Description The current auxiliary register and ARP contents are modified as specified, and control is passed to the designated program-memory address (pma). The pma can be either a symbolic or numeric address.

Words 2

Cycles

Cycles for a Single B Instruction

ROM	DARAM	SARAM	External
4	4	4	4+4p

Note: When this instruction reaches the execute phase of the pipeline, two additional instruction words have entered the pipeline. When the PC discontinuity is taken, these two instruction words are discarded.

Example

```
B       191,*+,AR1
```

The value 191 is loaded into the program counter, and the program continues to execute from that location. The current auxiliary register is incremented by 1, and ARP is set to point to auxiliary register 1 (AR1).

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) → PC

Status Bits None

Description Control is passed to the 16-bit address residing in the lower half of the accumulator.

Words 1

Cycles

Cycles for a Single BACC Instruction

ROM	DARAM	SARAM	External
4	4	4	4+3p

Note: When this instruction reaches the execute phase of the pipeline, two additional instruction words have entered the pipeline. When the 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, and the program continues to execute from that location.

Syntax **BANZ** *pma* [, *ind* [, **AR***n*]] Indirect addressing

Operands

pma: 16-bit program-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

BANZ *pma* [, *ind* [,**AR***n*]]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

If (current AR) $\neq$ 0
Then pma $\rightarrow$ PC
Else (PC) + 2 $\rightarrow$ PC
Modify (current AR) and (ARP) as specified

Status Bits None

Description Control is passed to the designated program-memory address (pma) if the contents of the current auxiliary register are not zero. Otherwise, control passes to the next instruction.The default modification to the current AR is a decrement by one. N loop iterations can be executed by initializing an auxiliary register (as a loop counter) to N–1 prior to loop entry. The pma can be either a symbolic or a numeric address.

Words 2

Cycles

Cycles for a Single BANZ Instruction

Condition	ROM	DARAM	SARAM	External
True	4	4	4	4+4p
False	2	2	2	2+2p

Note: The ’C20x performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken, these two instruction words are discarded.

Example 1

```
BANZ      PGM0      ;(PGM0 labels program address 0)
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	5h	AR0	4h

Because the content of AR0 is not zero, the program address denoted by PGM0 is loaded into the program counter (PC), and the program continues executing from that location. The default auxiliary register operation is a decrement of the current auxiliary register content; thus, AR0 contains 4h at the end of the execution.

or

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	0h	AR0	FFFFh

Because the content of AR0 is zero, the branch is not executed; instead, the PC is incremented by 2, and execution continues with the instruction following the BANZ instruction. Because of the default decrement, AR0 is decremented by 1, becoming –1.

Example 2

```
        MAR *,AR0          ;Set ARP to point to AR0.
        LAR AR1,#3         ;Load AR1 with 3.
        LAR AR0,#60h       ;Load AR0 with 60h.
PGM191  ADD *+,AR1         ;Loop: While AR1 not zero,
        BANZ PGM191,*-AR0  ;add data referenced by AR0
                           ;to accumulator and increment
                           ;AR0 value.
```

The contents of data-memory locations 60h–63h are added to the accumulator.

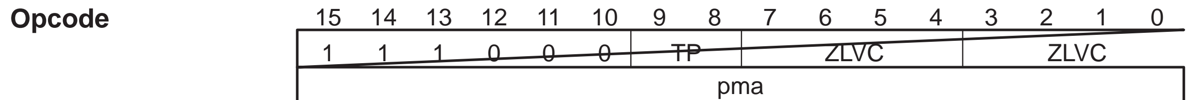
Syntax **BCND** *pma*, *cond* 1 [,*cond* 2] [,...]

Operands

pma: 16-bit program-memory address

cond	*Condition*
EQ	ACC = 0
NEQ	ACC ≠ 0
LT	ACC < 0
LEQ	ACC ≤ 0
GT	ACC > 0
GEQ	ACC ≥ 0
NC	C = 0
C	C = 1
NOV	OV = 0
OV	OV = 1
BIO	$\overline{\text{BIO}}$ low
NTC	TC = 0
TC	TC = 1
UNC	Unconditionally

Opcode



Note: The TP and ZLVC fields are defined on pages 7-3 and 7-4.

Execution

If *cond* 1 AND *cond* 2 AND ...
Then pma → PC
Else increment PC

Status Bits None

Description A branch is taken to the specified program-memory address (pma) if the specified conditions are met. Not all combinations of conditions are meaningful. For example, testing for LT and GT is contradictory. In addition, testing $\overline{\text{BIO}}$ is mutually exclusive to testing TC.

Words 2

Cycles

Cycles for a Single BCND Instruction

Condition	ROM	DARAM	SARAM	External
True	4	4	4	4+4p
False	2	2	2	2+2p

Note: The ’C20x performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken, these two instruction words are discarded.

Example

```
BCND    PGM191,LEQ,C
```

If the accumulator contents are less than or equal to zero and the carry bit is set, program address 191 is loaded into the program counter, and the program continues to execute from that location. If these conditions do not hold, execution continues from location PC + 2.

Syntax

BIT *dma*, *bit code*	Direct addressing
BIT *ind*, *bit code* [, **AR***n*]	Indirect addressing

Operands

dma:	7 LSBs of the data-memory address
bit code:	Value from 0 to 15 indicating which bit to test (see Figure 7–1)
n:	Value from 0 to 7 designating the next auxiliary register
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

BIT *dma*, *bit code*

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

BIT *ind*, *bit code* [,**AR***n*]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	bit code				1	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data bit number (15 – bit code)) → TC

Status Bits

Affects
TC

Description

The BIT instruction copies the specified bit of the data-memory value to the TC bit of status register ST1. Note that the BITT, CMPR, LST #1, and NORM instructions also affect the TC bit in ST1. A bit code value is specified that corresponds to a certain bit number of the data-memory value, as shown in Figure 7–1. For example, if you want to copy bit 6, you specify the bit code as 9, which is 15 minus six (15–6).

Figure 7–1. Bit Numbers and Their Corresponding Bit Codes for BIT Instruction

Bit code	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSB							Data-memory value								LSB

Words

1

Cycles

Cycles for a Single BIT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of a BIT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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, then set ARP = 1
```

	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

BITT *dma* — Direct addressing
BITT *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

BITT *dma*

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						

BITT *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data bit number (15 –TREG(3:0))) → TC

Status Bits

Affects
TC

Description

The BITT instruction copies the specified bit of the data-memory value to the TC bit of status register ST1. Note that the BITT, CMPR, LST #1, and NORM instructions also affect the TC bit in status register ST1. The bit number is specified by a bit code value contained in the four LSBs of the TREG, as shown in Figure 7–2.

Figure 7–2. Bit Numbers and Their Corresponding Bit Codes for BITT Instruction

Bit code (in 4 LSBs of TREG)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data-memory value	MSB															LSB

Words

1

Cycles

Cycles for a Single BITT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an BITT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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
TREG	1h	TREG	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
TREG	0Eh	TREG	0Eh
TC	0	TC	0

Syntax

General syntax: **BLDD** *source, destination*

BLDD #*lk*, *dma*	Direct with long immediate source
BLDD #*lk*, *ind* [, **AR***n*]	Indirect with long immediate source
BLDD *dma*, **#***lk*	Direct with long immediate destination
BLDD *ind*, **#***lk* [, **AR***n*]	Indirect with long immediate destination

Operands

dma:	7 LSBs of the data-memory address
n:	Value from 0 to 7 designating the next auxiliary register
lk:	16-bit long immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

BLDD #*lk*, *dma*

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						
lk															

BLDD #*lk*, *ind* [, **AR***n*]

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	ARU			N	NAR		
lk															

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

BLDD *dma*, **#***lk*

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						
lk															

BLDD *ind*, **#***lk* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(PC) → MSTACK
lk → PC
(source) → destination
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC

While (repeat counter) ≠ 0:
  (source) → destination
  For indirect, modify (current AR) and (ARP) as specified
  (PC) + 1 → PC
  (repeat counter) –1 → repeat counter

(MSTACK) → PC

Status Bits

None

Description

The word in data memory pointed to by *source* is copied to a data-memory space pointed to by *destination*. The word of the source and/or destination space can be pointed to with a long-immediate value or by a data-memory address. Note that not all source/destination combinations of pointer types are valid.

Note:

BLDD will not work with memory-mapped registers.

RPT can be used 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 (RPTC) at the beginning of the instruction. When the BLDD instruction is repeated, the source (destination) address specified by the long immediate constant is stored to the PC. Because the PC is incremented by 1 during each repetition, it is possible to access a series of source (destination) addresses. If you use indirect addressing to specify the destination (source) address, a new destination (source) address can be accessed during each repetition. If you use the direct addressing mode, the specified destination (source) address is a constant; it will not be modified during each repetition.

The source and destination blocks do not have to be entirely on chip or off chip. Interrupts are inhibited during a BLDD operation used with the RPT instruction. When used with RPT, BLDD becomes a single-cycle instruction once the RPT pipeline is started.

Words

2

Cycles

Cycles for a Single BLDD Instruction

Operand	ROM	DARAM	SARAM	External
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
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	$4+d_{src}+d_{dst}$	$4+d_{src}+d_{dst}$	$4+d_{src}+d_{dst}$	$6+d_{src}+d_{dst}+2p$

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

Cycles for a Repeat (RPT) Execution of a BLDD Instruction

Operand	ROM	DARAM	SARAM	External
Source: DARAM Destination: DARAM	n+2	n+2	n+2	n+2+2p
Source: SARAM Destination: DARAM	n+2	n+2	n+2	n+2+2p
Source: External Destination: DARAM	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}+2p$
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	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}$ $n+4+nd_{src}$†	$n+2+nd_{src}+2p$
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	Data Memory 300h	0h
320h	0Fh	320h	0h

Example 2

```
BLDD      *+,#321h,AR3
```

	Before Instruction		After Instruction
ARP	2	ARP	3
AR2	301h	AR2	302h
Data Memory 301h	01h	Data Memory 301h	01h
321h	0Fh	321h	01h

Syntax

General syntax: **BLPD** *source*, *destination*

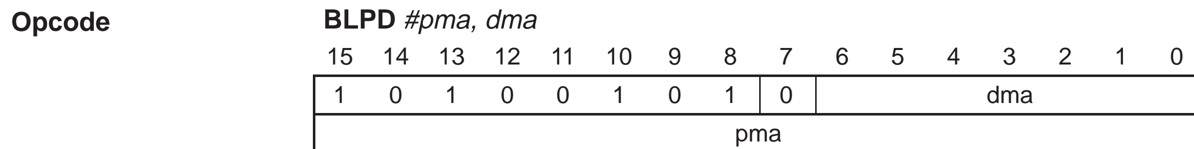
BLPD #*pma*, *dma*	Direct with long immediate source
BLPD #*pma*, *ind* [, **AR***n*]	Indirect with long immediate source

Operands

pma:	16-bit program-memory address
dma:	7 LSBs of the data-memory address
n:	Value from 0 to 7 designating the next auxiliary register
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

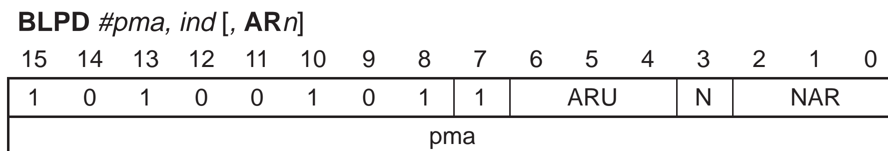
Opcode

BLPD *#pma, dma*



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						
pma															

BLPD *#pma, ind* [*,* **AR***n*]



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	ARU			N	NAR		
pma															

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(PC) → MSTACK
pma → PC
(source) → destination
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC

While (repeat counter) ≠ 0:
  (source) → destination
  For indirect, modify (current AR) and (ARP) as specified
  (PC) + 1 → PC
  (repeat counter) –1 → repeat counter

(MSTACK) → PC

Status Bits

None

Description

A word in program memory pointed to by the *source* is copied to data-memory space pointed to by *destination*. The first word of the source space is pointed to by a long-immediate value. The data-memory destination space is pointed to by a data-memory address or auxiliary register pointer. Not all source/destination combinations of pointer types are valid.

RPT can be used with the BLPD instruction to move consecutive words. The number of words to be moved is one greater than the number contained in the repeat counter (RPTC) at the beginning of the instruction. When the BLPD instruction is repeated, the source (program-memory) address specified by the long immediate constant is stored to the PC. Because the PC is incremented by 1 during each repetition, it is possible to access a series of program-memory addresses. If you use indirect addressing to specify the destination (data-memory) address, a new data-memory address can be accessed during each repetition. If you use the direct addressing mode, the specified data-memory address is a constant; it will not be modified during each repetition.

The source and destination blocks do not have to be entirely on chip or off chip. Interrupts are inhibited during a repeated BLPD instruction. When used with RPT, BLPD becomes a single-cycle instruction once the RPT pipeline is started.

Words

2

Cycles

Cycles for a Single BLPD Instruction

Operand	ROM	DARAM	SARAM	External
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†	$3+2p_{code}$
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 of a BLPD Instruction

Operand	ROM	DARAM	SARAM	External
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†	$n+2+2p_{code}$

† 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 of a BLPD Instruction (Continued)

Operand	ROM	DARAM	SARAM	External
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

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

Description The current program counter (PC) is incremented and pushed onto the top of the stack (TOS). Then, the contents of the lower half of the accumulator are loaded into the PC. Execution continues at this address.

The CALA instruction is used to perform computed subroutine calls.

Words 1

Cycles

Cycles for a Single CALA Instruction

ROM	DARAM	SARAM	External
4	4	4	4+3p

Note: When this instruction reaches the execute phase of the pipeline, two additional instruction words have entered the pipeline. When the 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

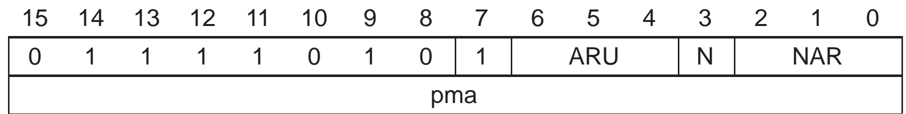
Syntax **CALL** *pma* [, *ind* [, **AR***n*]] Indirect addressing

Operands

pma: 16-bit program-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

CALL *pma* [, *ind* [, **AR***n*]]



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	ARU			N	NAR		
pma															

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

PC + 2 → TOS
pma → PC
Modify (current AR) and (ARP) as specified.

Status Bits None

Description The current program counter (PC) is incremented and pushed onto the top of the stack (TOS). Then, the contents of the pma, either a symbolic or numeric address, are loaded into the PC. Execution continues at this address. The current auxiliary register and ARP contents are modified as specified.

Words 2

Cycles

Cycles for a Single CALL Instruction

ROM	DARAM	SARAM	External
4	4	4	4+4p†

Note: When this instruction reaches the execute phase of the pipeline, two additional instruction words have entered the pipeline. When the PC discontinuity is taken, these two instruction words are discarded.

Example

```
CALL     191,*+,AR0
```

	Before Instruction		After Instruction
ARP	1	ARP	0
AR1	05h	AR1	06h
PC	30h	PC	0BFh
TOS	100h	TOS	32h

Program address 0BFh (191) is loaded into the program counter, and the program continues executing from that location.

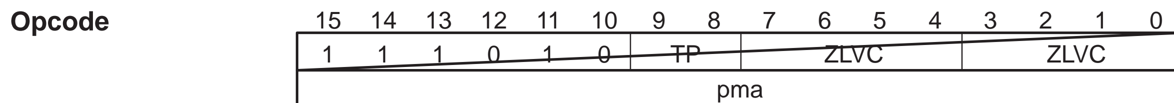
Syntax **CC** *pma*, *cond* 1 [,*cond* 2] [,...]

Operands

pma: 16-bit program-memory address

cond	*Condition*
EQ	ACC = 0
NEQ	ACC ≠ 0
LT	ACC < 0
LEQ	ACC ≤ 0
GT	ACC > 0
GEQ	ACC ≥ 0
NC	C = 0
C	C = 1
NOV	OV = 0
OV	OV = 1
BIO	$\overline{BIO}$ low
NTC	TC = 0
TC	TC = 1
UNC	Unconditionally

Opcode



Note: The TP and ZLVC fields are defined on pages 7-3 and 7-4.

Execution

If *cond* 1 AND *cond* 2 AND ...
Then
PC + 2 → TOS
pma → PC
Else
Increment PC

Status Bits None

Description Control is passed to the specified program-memory address (pma) if the specified conditions are met. Not all combinations of conditions are meaningful. For example, testing for LT and GT is contradictory. In addition, testing $\overline{BIO}$ is mutually exclusive to testing TC. The CC instruction operates like the CALL instruction if all conditions are true.

Words 2

Cycles

Cycles for a Single CC Instruction

Condition	ROM	DARAM	SARAM	External
True	4	4	4	4+4p†
False	2	2	2	2+2p

† The processor performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken these two instruction words are discarded.

Example

```
CC        PGM191,LEQ,C
```

If the accumulator contents are less than or equal to zero and the carry bit is set, 0BFh (191) is loaded into the program counter, and the program continues to execute from that location. If the conditions are not met, execution continues at the instruction following the CC instruction.

Syntax **CLRC** *control bit*

Operands control bit: Select one of the following control bits:

C	Carry bit of status register ST1
CNF	RAM configuration control bit of status register ST1
INTM	Interrupt mode bit of status register ST0
OVM	Overflow mode bit of status register ST0
SXM	Sign-extension mode bit of status register ST1
TC	Test/control flag bit of status register ST1
XF	XF pin status bit of status register ST1

Opcode

CLRC C

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

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

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 OVM

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

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 TC

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 XF

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 Increment PC, then ...
$0 \rightarrow$ control bit

Status Bits None

Description The specified control bit is cleared to 0. Note that the LST instruction can also be used to load ST0 and ST1. See section 3.5, *Status Registers ST0 and ST1* on page 3-15, for more information on each of these control bits.

Words 1

Cycles

Cycles for a Single CLRC Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of a CLRC Instruction

ROM	DARAM	SARAM	External
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 Increment PC, then ...
$\overline{(ACC)} \rightarrow ACC$

Status Bits None

Description The contents of the accumulator are replaced with its logical inversion (1s complement). The carry bit is unaffected.

Words 1

Cycles

Cycles for a Single CMPL Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an CMPL Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example `CMPL`

Syntax **CMPR** *CM*

Operands CM: Value from 0 to 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	

Execution Increment PC, then ...
Compare (current AR) to (AR0) and place the result in the TC bit of status register ST1.

Status Bits *Affects*
TC

This instruction is not affected by SXM. It does not affect SXM.

Description The CMPR instruction performs a comparison specified by the value of CM:

If CM = 00, test whether (current AR) = (AR0)
If CM = 01, test whether (current AR) < (AR0)
If CM = 10, test whether (current AR) > (AR0)
If CM = 11, test whether (current AR) ≠ (AR0)

If the condition is true, the TC bit is set to 1. If the condition is false, the TC bit is cleared to 0.

Note that the auxiliary register values are treated as unsigned integers in the comparisons.

Words 1

Cycles

Cycles for a Single CMPR Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an CMPR Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
CMPR    2        ;(current AR) > (AR0)?
```

	Before Instruction		After Instruction
ARP	4	ARP	4
AR0	0FFFFh	AR0	0FFFFh
AR4	7FFFh	AR4	7FFFh
TC	1	TC	0

Syntax

DMOV *dma*	Direct addressing
DMOV *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

DMOV *dma*

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						

DMOV *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → data-memory address + 1

Status Bits

Affected by
CNF

Description

The contents of the specified data-memory address are copied into the contents of the next higher address. When data is copied from the addressed location to the next higher location, the contents of the addressed location remain unaltered.

DMOV works only within on-chip data DARAM blocks. It works within any configurable RAM block if that block is configured as data memory. In addition, the data move function is continuous across block boundaries. The data move function cannot be performed on external data memory. If the instruction specifies an external memory address, DMOV reads the specified memory location but performs *no* operations.

The data move function is useful in implementing the z^{-1} delay encountered in digital signal processing. The DMOV function is a subtask of the LTD and MACD instructions (see the LTD and MACD instructions for more information).

Words

1

Cycles

Cycles for a Single DMOV Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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
‡ If used on external memory, DMOV reads the specified memory location but performs no operations.

Cycles for a Repeat (RPT) Execution of a DMOV Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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
‡ If used on external memory, DMOV reads the specified memory location but performs no operations.

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
AR0	30Ah	AR0	30Ah
Data Memory 30Ah	40h	Data Memory 30Ah	40h
Data Memory 30Bh	41h	Data Memory 30Bh	40h

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 Increment PC, then wait for unmasked or nonmaskable hardware interrupt.

Status Bits *Affected by*
INTM

Description The IDLE instruction forces the program being executed to halt until the CPU receives a request from an unmasked hardware interrupt (external or internal), $\overline{\text{NMI}}$, or reset. Execution of the IDLE instruction causes the ’C20x to enter a power-down mode. The PC is incremented once before the ’C20x enters power down; it is not incremented during the idle state. On-chip peripherals remain active; thus, their interrupts are among those that can wake the processor.

The idle state is exited by an unmasked interrupt even if INTM is 1. (INTM, the interrupt mode bit of status register ST0, normally disables maskable interrupts when it is set to 1.) When the idle state is exited by an unmasked interrupt, the CPU’s next action, however, depends on INTM:

- If INTM is 0, the program branches to the corresponding interrupt service routine.
- If INTM is 1, the program continues executing at the instruction following the IDLE.

$\overline{\text{NMI}}$ and reset are not maskable; therefore, if the idle state is exited by $\overline{\text{NMI}}$ or reset, the corresponding interrupt service routine will be executed, regardless of INTM.

Words 1

Cycles

Cycles for a Single IDLE Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Example

```
IDLE        ;The processor idles until a hardware reset,
            ;a hardware NMI, or an unmasked interrupt
            ;occurs.
```

Syntax

IN *dma*, *PA* — Direct addressing
IN *ind*, *PA* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
PA: 16-bit I/O port or I/O-mapped register address
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

IN *dma , PA*

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

IN *ind ,PA* [,**AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
PA → address bus lines A15–A0
Data bus lines D15–D0 → data-memory address
(PA) → data-memory address

Status Bits

None

Description

The IN instruction reads a 16-bit value from an I/O location into the specified data-memory location. The $\overline{\text{IS}}$ line goes low to indicate an I/O access. The $\overline{\text{STRB}}$, $\overline{\text{RD}}$, and READY timings are the same as for an external data-memory read.

The repeat (RPT) instruction can be used with the IN instruction to read in consecutive words from I/O space to data space.

Words

2

Cycles

Cycles for a Single IN Instruction

Operand	Program			
	ROM	DARAM	SARAM	External
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 operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution of an IN Instruction

Operand	Program			
	ROM	DARAM	SARAM	External
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 operand and the code are in the same SARAM block

Example 1

```
IN    7,1000h        ;Read in word from peripheral on
                     ;port address 1000h. Store word in
                     ;data memory location 307h (DP=6).
```

Example 2

```
IN    *,5h           ;Read in word from peripheral on
                     ;port address 5h. Store word in
                     ;data memory location specified by
                     ;current auxiliary register.
```

Syntax **INTR** *K*

Operands K: Value from 0 to 31 that indicates the interrupt vector location to branch to

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			K		

Execution (PC) + 1 → stack
corresponding interrupt vector location → PC

Status Bits *Affects*
INTM

This instruction is not affected by INTM.

Description The processor has locations for 32 interrupt vectors; each location is represented by a value K from 0 to 31. The INTR instruction is a software interrupt that transfers program control to the program-memory address specified by K. The vector at that address then leads to the corresponding interrupt service routine. Thus, the instruction allows any one of the interrupt service routines to be executed from your software. For a list of interrupts and their corresponding K values, see section 5.6.2, *Interrupt Table*, on page 5-16. During execution of the instruction, the value PC + 1 (the return address) is pushed onto the stack. Neither the INTM bit nor the interrupt masks affect the INTR instruction. An INTR for the external interrupts looks exactly like an external interrupt (an interrupt acknowledge is generated, and maskable interrupts are globally disabled by setting INTM = 1).

Words 1

Cycles

Cycles for a Single INTR Instruction

ROM	DARAM	SARAM	External
4	4	4	4+3p†

† The processor performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken, these two instruction words are discarded.

Example

```
INTR      3         ;PC + 1 is pushed onto the stack.
                    ;Then control is passed to program
                    ;memory location 6h.
```

Execution Increment PC, then ...

Event	*Addressing mode*
(data-memory address) $\times 2^{shift} \rightarrow$ ACC	Direct or indirect
(data-memory address) $\times 2^{16} \rightarrow$ ACC	Direct or indirect (shift of 16)
lk $\times 2^{shift} \rightarrow$ ACC	Long immediate

Status Bits

Affected by
SXM

Description

The contents of the specified data-memory address or a 16-bit constant are left shifted and loaded into the accumulator. During shifting, low-order bits are zero filled. High-order bits are sign extended if SXM = 1 and zeroed if SXM = 0.

Words

Words	*Addressing mode*
1	Direct or indirect
2	Long immediate

Cycles

Cycles for a Single LACC Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LACC Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 LACC Instruction (Using Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+2p

Example 1

```
LACC    6,4     ;(DP = 8: addresses 0400h–047Fh,
                ;SXM = 0)
```

	Before Instruction		After Instruction
Data Memory 406h	01h	Data Memory 406h	01h
ACC (C = X)	012345678h	ACC (C = X)	10h

Example 2

```
LACC    *,4     ;(SXM = 0)
```

	Before Instruction		After Instruction
ARP	2	ARP	2
AR2	0300h	AR2	0300h
Data Memory 300h	0FFh	Data Memory 300h	0FFh
ACC (C = X)	12345678h	ACC (C = X)	0FF0h

Example 3

```
LACC    #0F000h,1 ;(SXM = 1)
```

	Before Instruction		After Instruction
ACC (C = X)	012345678h	ACC (C = X)	FFFFE000h

Syntax

LACC *dma* [, *shift*]	Direct addressing
LACC *dma,* **16**	Direct with left shift of 16
LACC *ind* [, *shift* [, **AR***n*]]	Indirect addressing
LACC *ind*, **16**[, **AR***n*]	Indirect with left shift of 16
LACC #*lk* [, *shift*]	Long immediate addressing

Operands

dma:	7 LSBs of the data-memory address
shift:	Left shift value from 0 to 15 (defaults to 0)
n:	Value from 0 to 7 designating the next auxiliary register
lk:	16-bit long immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LACC *dma* [, *shift*]

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

LACC *dma*, **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						

LACC *ind* [, *shift*[, **AR***n*]]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

LACC *ind*, **16**[, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

LACC #*lk* [, *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	shift			
lk															

Syntax

LACL *dma*	Direct addressing
LACL *ind* [, **AR***n*]	Indirect addressing
LACL #*k*	Short immediate

Operands

dma:	7 LSBs of the data-memory address
n:	Value from 0 to 7 designating the next auxiliary register
k:	8-bit short immediate value
ind:	Select one of the following seven options: * *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LACL *dma*

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						

LACL *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

LACL #*k*

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

Execution

Increment PC, then ...

Events	*Addressing mode*
0 → ACC(31:16) (data-memory address) → ACC(15:0)	Direct or indirect
0 → ACC(31:8) k → ACC(7:0)	Short immediate

Status Bits

This instruction is not affected by SXM.

Description

The contents of the addressed data-memory location or a zero-extended 8-bit constant are loaded into the 16 low-order bits of the accumulator. The upper half of the accumulator is zeroed. The data is treated as an unsigned 16-bit number rather than a 2s-complement number. There is no sign extension of the operand with this instruction, regardless of the state of SXM.

Words

1

Cycles

Cycles for a Single LACL Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LACL Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 LACL Instruction (Using Immediate Addressing)

ROM	DARAM	SARAM	External
1	1	1	1+p

Example 1

```
LACL    1         ;(DP = 6: addresses 0300h–037Fh)
```

		Before Instruction			After Instruction
Data Memory 301h		0h	Data Memory 301h		0h
ACC	X (C)	7FFFFFFFh	ACC	X (C)	0h

Example 2

```
LACL    *–,AR4
```

		Before Instruction			After Instruction
ARP		0	ARP		4
AR0		401h	AR0		400h
Data Memory 401h		00FFh	Data Memory 401h		00FFh
ACC	X (C)	7FFFFFFFh	ACC	X (C)	0FFh

Example 3

```
LACL    #10h
```

Syntax

LACT *dma*	Direct addressing
LACT *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LACT *dma*

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						

LACT *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) $\times$ $2^{(TREG(3:0))}$ $\rightarrow$ ACC

If SXM = 1:
Then (data-memory address) is sign extended.
If SXM = 0:
Then (data-memory address) is not sign extended.

Status Bits

Affected by
SXM

Description

The LACT instruction loads the accumulator with a data-memory value that has been left shifted. The left shift is specified by the four LSBs of the TREG, resulting in shift options from 0 to 15 bits. Using the four LSBs of the TREG as a shift code provides a dynamic shift mechanism. During shifting, the high-order bits are sign extended if SXM = 1 and zeroed if SXM = 0.

LACT may be used to denormalize a floating-point number if the actual exponent is placed in the four LSBs of the TREG register and the mantissa is referenced by the data-memory address. This method of denormalization can be used only when the magnitude of the exponent has four bits or less.

Words

1

Cycles

Cycles for a Single LACT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LACT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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    1         ;(DP = 6: addresses 0300h–037Fh,
                  ;SXM = 0)
```

		Before Instruction			After Instruction
Data Memory 301h		1376h	Data Memory 301h		1376h
TREG		14h	TREG		14h
ACC	X (C)	98F7EC83h	ACC	X (C)	13760h

Example 2

```
LACT    *–,AR3    ;(SXM = 1)
```

		Before Instruction			After Instruction
ARP		1	ARP		3
AR1		310h	AR1		30Fh
Data Memory 310h		0FF00h	Data Memory 310h		0FF00h
TREG		11h	TREG		11h
ACC	X (C)	098F7EC83h	ACC	X (C)	0FFFFFE00h

Syntax

LAR AR*x*, *dma*	Direct addressing
LAR AR*x*, *ind* [, **AR***n*]	Indirect addressing
LAR AR*x*, **#***k*	Short immediate addressing
LAR AR*x*, **#***lk*	Long immediate addressing

Operands

x:	Value from 0 to 7 designating the auxiliary register to be loaded
dma:	7 LSBs of the data-memory address
k:	8-bit short immediate value
lk:	16-bit long immediate value
n:	Value from 0 to 7 designating the next auxiliary register
ind:	Select one of the following seven options: * *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LAR AR*x*, *dma*

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

LAR AR*x*, *ind* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

LAR AR*x*, **#***k*

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

LAR AR*x*, **#***lk*

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	1	x		
lk															

Execution

Increment PC, then ...

Event	*Addressing mode*
(data-memory address) → ARx	Direct or indirect
k → ARx	Short immediate
lk → ARx	Long immediate

Status Bits None

Description

The contents of the specified data-memory address or an 8-bit or 16-bit constant are loaded into the specified auxiliary register (ARx). The specified constant is treated as an unsigned integer, regardless of the value of SXM.

The LAR and SAR (store auxiliary register) instructions can be used to load and store the auxiliary registers during subroutine calls and interrupts. If an auxiliary register 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.

Words

Words	Addressing mode
1	Direct, indirect or short immediate
2	Long immediate

Cycles

Cycles for a Single LAR Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
DARAM	2	2	2	$2+p_{code}$
SARAM	2	2	2, 3†	$2+p_{code}$
External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

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

Cycles for a Repeat (RPT) Execution of an LAR Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
DARAM	2n	2n	2n	$2n+p_{code}$
SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}p_{code}$

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

Cycles for a Single LAR Instruction (Using Short Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	$2+p_{code}$

Cycles for a Single LAR Instruction (Using Long Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+2p

Example 1

```
LAR     AR0,16    ;(DP = 6: addresses 0300h–037Fh)
```

	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	0FF09h	AR4	01h

Example 4

```
LAR     AR6,#3FFFh
```

	Before Instruction		After Instruction
AR6	0h	AR6	3FFFh

Syntax

LDP *dma*	Direct addressing
LDP *ind* [, **AR***n*]	Indirect addressing
LDP #*k*	Short immediate addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
k: 9-bit short immediate value
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LDP *dma*

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						

LDP *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

LDP #*k*

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

Execution

Increment PC, then ...

Event	*Addressing mode*
Nine LSBs of (data-memory address) → DP	Direct or indirect
k → DP	Short immediate

Status Bits

Affects
DP

Description

The nine LSBs of the contents of the addressed data-memory location or a 9-bit immediate value is loaded into the data page pointer (DP) of status register ST0. The DP can also be loaded by the LST instruction.

In direct addressing, the 9-bit DP and the 7-bit value specified in the instruction (dma) are concatenated to form the 16-bit data-memory address accessed by the instruction. The DP provides the 9 MSBs, and dma provides the 7 LSBs.

Words

1

Cycles

Cycles for a Single LDP Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
DARAM	2	2	2	2+p_{code}
SARAM	2	2	2, 3†	2+p_{code}
External	2+d_{src}	2+d_{src}	2+d_{src}	3+d_{src}+p_{code}

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

Cycles for a Repeat (RPT) Execution of an LDP Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
DARAM	2n	2n	2n	2n+p_{code}
SARAM	2n	2n	2n, 2n+1†	2n+p_{code}
External	2n+nd_{src}	2n+nd_{src}	2n+nd_{src}	2n+1+n$d_{src}$$p_{code}$

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

Cycles for a Single LDP Instruction (Using Short Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+p_{code}

Example 1

```
LDP       127        ;(DP = 511: addresses FF80h–FFFFh)
```

	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

LPH *dma* — Direct addressing
LPH *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LPH *dma*

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						

LPH *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → PREG (31:16)

Status Bits

None

Description

The 16 high-order bits of the PREG are loaded with the content of the specified data-memory address. The low-order PREG bits are unaffected.

The LPH instruction can be used for restoring the high-order bits of the PREG after interrupts and subroutine calls.

Words

1

Cycles

Cycles for a Single LPH Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LPH Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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	0F79Ch	Data Memory 200h	0F79Ch
PREG	30079844h	PREG	0F79C9844h

Example 2

```
LPH      *,AR6
```

	Before Instruction		After Instruction
ARP	5	ARP	6
AR5	200h	AR5	200h
Data Memory 200h	0F79Ch	Data Memory 200h	0F79Ch
PREG	30079844h	PREG	0F79C9844h

Syntax

LST #*m*, *dma*	Direct addressing
LST #*m*, *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
m: Select one of the following:
  0 Indicates that ST0 will be loaded
  1 Indicates that ST1 will be loaded
ind: Select one of the following seven options:
  * *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LST #0, *dma*

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						

LST #0, *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

LST #1, *dma*

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						

LST #1, *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

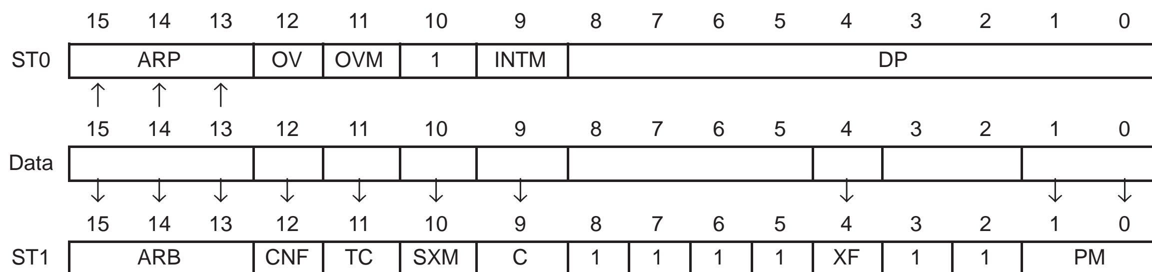
Increment PC, then ...
(data-memory address) → status register STm

For details about the differences between an LST #0 operation and an LST #1 operation, see Figure 7–3, Figure 7–4, and the description category below.

Figure 7–3. LST #0 Operation

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data	↓	↓	↓	↓	↓			↓	↓	↓	↓	↓	↓	↓	↓	↓
ST0	ARP			OV	OVM	1	INTM	DP								

Figure 7–4. LST #1 Operation



Status Bits

<u>Affects</u>
ARB, ARP, OV, OVM, DP, CNF, TC, SXM, C, XF, and PM

This instruction does not affect INTM.

Description

The specified status register (ST0 or ST1) is loaded with the addressed data-memory value. Note the following points:

- The LST #0 operation does not affect the ARB field in the ST1 register, even though a new ARP is loaded.
- During the LST #1 operation, the value loaded into ARB is also loaded into ARP.
- If a next AR value is specified as an operand in the indirect addressing mode, this operand is ignored. ARP is loaded with the three MSBs of the value contained in the addressed data-memory location.
- Reserved bit values in the status registers are always read as 1s. Writes to these bits have no effect.

The LST instruction can be used for restoring the status registers after subroutine calls and interrupts.

Words 1

Cycles

Cycles for a Single LST Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
DARAM	2	2	2	$2+p_{code}$
SARAM	2	2	2, 3†	$2+p_{code}$
External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

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

Cycles for a Repeat (RPT) Execution of an LST Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
DARAM	2n	2n	2n	$2n+p_{code}$
SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the 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. Also note that
                 ;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	05ECh	ST1	05ECh

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	EE00h	ST0	EE04h
ST1	F7ECh	ST1	F7ECh

Example 4

```
LST     #1,00h    ;(DP = 6)
                  ;Note that the ARB is loaded with
                  ;the new ARP value.
```

	Before Instruction		After Instruction
Data Memory 300h	E1BCh	Data Memory 300h	E1BCh
ST0	0406h	ST0	E406h
ST1	09ECh	ST1	E1FCh

Syntax

LT *dma* — Direct addressing
LT *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LT *dma*

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						

LT *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → TREG

Status Bits

None

Description

TREG is loaded with the contents of the specified data-memory address. The LT instruction may be used to load TREG in preparation for multiplication. See also the LTA, LTD, LTP, LTS, MPY, MPYA, MPYS, and MPYU instructions.

Words

1

Cycles

Cycles for a Single LT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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    24            ;(DP = 8: addresses 0400h–047Fh)
```

	Before Instruction		After Instruction
Data Memory 418h	62h	Data Memory 418h	62h
TREG	3h	TREG	62h

Example 2

```
LT    *,AR3
```

	Before Instruction		After Instruction
ARP	2	ARP	3
AR2	418h	AR2	418h
Data Memory 418h	62h	Data Memory 418h	62h
TREG	3h	TREG	62h

Syntax

LTA *dma* — Direct addressing
LTA *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LTA *dma*

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						

LTA *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → TREG
(ACC) + shifted (PREG) → ACC

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

Description

TREG is loaded with the contents of the specified data-memory address. The contents of the product register, shifted as defined by the PM status bits, are added to the accumulator, and the result is placed in the accumulator.

The carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.

The function of the LTA instruction is a subtask of the LTD instruction.

Words

1

Cycles

Cycles for a Single LTA Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LTA Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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     36      ;(DP = 6: addresses 0300h–037Fh,
                ;PM =0: no shift of product)
```

	Before Instruction		After Instruction
Data Memory 324h	62h	Data Memory 324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC	C: X, 5h	ACC	C: 0, 14h

Example 2

```
LTA     *,AR5   ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	4	ARP	5
AR4	324h	AR4	324h
Data Memory 324h	62h	Data Memory 324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC	C: X, 5h	ACC	C: 0, 14h

Syntax

LTD *dma* — Direct addressing
LTD *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LTD *dma*

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						

LTD *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → TREG
(data-memory address) → data-memory address + 1
(ACC) + shifted (PREG) → ACC

Status Bits

Affected by	Affects
PM and OVM	C and OV

Description

TREG is loaded with the contents of the specified data-memory address. The contents of the PREG, shifted as defined by the PM status bits, are added to the accumulator, and the result is placed in the accumulator. The contents of the specified data-memory address are also copied to the next higher data-memory address.

This instruction is valid for all blocks of on-chip RAM configured as data memory. The data move function is continuous across the boundaries of contiguous blocks of memory but cannot be used with external data memory or memory-mapped registers. The data move function is described under the instruction DMOV.

Note:

If LTD is used with external data memory, its function is identical to that of LTA; that is, the previous product will be accumulated, and the TREG will be loaded from external data memory, but *the data move will not occur.*

The carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.

Words 1

Cycles

Cycles for a Single LTD Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External‡**
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
‡ If the LTD instruction is used with external memory, the data move will not occur. (The previous product will be accumulated, and the TREG will be loaded.)

Cycles for a Repeat (RPT) Execution of an LTD Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External‡**
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
‡ If the LTD instruction is used with external memory, the data move will not occur. (The previous product will be accumulated, and the TREG will be loaded.)

Example 1

```
LTD     126     ;(DP = 7: addresses 0380h–03FFh,
                ;PM = 0: no shift of product).
```

	Before Instruction		After Instruction
Data Memory 3FEh	62h	Data Memory 3FEh	62h
Data Memory 3FFh	0h	Data Memory 3FFh	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC (C = X)	5h	ACC (C = 0)	14h

Example 2

```
LTD     *,AR3     ;(PM = 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
TREG		3h	TREG		62h
PREG		0Fh	PREG		0Fh
ACC	X (C)	5h	ACC	0 (C)	14h

Note: The data move function for LTD can occur only within on-chip data memory RAM blocks.

Syntax

LTP *dma* — Direct addressing
LTP *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LTP *dma*

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						

LTP *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → TREG
shifted (PREG) → ACC

Status Bits

Affected by
PM

Description

The TREG is loaded with the content of the addressed data-memory location, and the PREG value is stored in the accumulator. The shift at the output of the PREG is controlled by the PM status bits.

Words

1

Cycles

Cycles for a Single LTP Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LTP Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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       36         ;(DP = 6: addresses 0300h–037Fh,
                     ;PM = 0: no shift of product)
```

	Before Instruction		After Instruction
Data Memory 324h	62h	Data Memory 324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC (C = X)	5h	ACC (C = X)	0Fh

Example 2

```
LTP       *,AR5      ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	2	ARP	5
AR2	324h	AR2	324h
Data Memory 324h	62h	Data Memory 324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC (C = X)	5h	ACC (C = X)	0Fh

Syntax

LTS *dma* — Direct addressing
LTS *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

LTS *dma*

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						

LTS *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → TREG
ACC – shifted (PREG) → ACC

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

Description

TREG is loaded with the contents of the addressed data-memory location. The contents of the product register, shifted as defined by the contents of the PM status bits, are subtracted from the accumulator. The result is placed in the accumulator.

The carry bit is cleared (C = 0) if the result of the subtraction generates a borrow, and is set (C = 1) if it does not generate a borrow.

Words

1

Cycles for a Single LTS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an LTS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0300h–037Fh,
                    ;PM = 0: no shift of product)
```

	Before Instruction		After Instruction
Data Memory 324h	62h	Data Memory 324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC	C: X; 05h	ACC	C: 0; 0FFFFFFF6h

Example 2

```
LTS       *,AR2     ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	1	ARP	2
AR1	324h	AR1	324h
324h	62h	324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC	C: X; 05h	ACC	C: 0; 0FFFFFFF6h

Syntax

MAC *pma*, *dma*	Direct addressing
MAC *pma*, *ind* [, **AR***n*]	Indirect addressing

Operands

dma:	7 LSBs of the data-memory address
pma:	16-bit program-memory address
n:	Value from 0 to 7 designating the next auxiliary register
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

MAC *pma*, *dma*

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

MAC *pma*, *ind* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then . . .
(PC) → MSTACK
pma → PC
(ACC) + shifted (PREG) → ACC
(data-memory address) → TREG
(data-memory address) × (pma) → PREG
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC

While (repeat counter) ≠ 0:
 (ACC) + shifted (PREG) → ACC
 (data-memory address) → TREG
 (data-memory address) × (pma) → PREG
 For indirect, modify (current AR) and (ARP) as specified
 (PC) + 1 → PC
 (repeat counter) – 1 → repeat counter

(MSTACK) → PC

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

Description

The MAC instruction:

- ❑ Adds the previous product, shifted as defined by the PM status bits, to the accumulator. The carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.
- ❑ Loads the TREG with the content of the specified data-memory address.
- ❑ Multiplies the data-memory value in the TREG by the contents of the specified program-memory address.

The data and program memory locations on the ’C20x may be any nonreserved on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, the CNF bit must be set to 1.

When the MAC instruction is repeated, the program-memory address contained in the PC is incremented by 1 during each repetition. This makes it possible to access a series of operands in program memory. If you use indirect addressing to specify the data-memory address, a new data-memory address can be accessed during each repetition. If you use the direct addressing mode, the specified data-memory address is a constant; it will not be modified during each repetition.

MAC is useful for long sum-of-products operations because, when repeated, it becomes a single-cycle instruction once the RPT pipeline is started.

Words

2

Cycles

Cycles for a Single MAC Instruction

Operand	ROM	DARAM	SARAM	External
Operand 1: DARAM/ ROM Operand 2: DARAM	3	3	3	$3+2p_{code}$
Operand 1: SARAM Operand 2: DARAM	3	3	3	$3+2p_{code}$
Operand 1: External Operand 2: DARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
Operand 1: DARAM/ ROM Operand 2: SARAM	3	3	3	$3+2p_{code}$
Operand 1: SARAM Operand 2: SARAM	3 4†	3 4†	3 4†	$3+2p_{code}$ $4+2p_{code}$†
Operand 1: External Operand 2: SARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
Operand 1: DARAM/ ROM Operand 2: External	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
Operand 1: SARAM Operand 2: External	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
Operand 1: External Operand 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 of an MAC Instruction

Operand	ROM	DARAM	SARAM	External
Operand 1: DARAM/ ROM Operand 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Operand 1: SARAM Operand 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Operand 1: External Operand 2: DARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$

† If both operands are in the same SARAM block

Cycles for a Repeat (RPT) Execution of an MAC Instruction (Continued)

Operand	ROM	DARAM	SARAM	External
Operand 1: DARAM/ ROM Operand 2: SARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Operand 1: SARAM Operand 2: SARAM	n+2 2n+2†	n+2 2n+2†	n+2 2n+2†	$n+2+2p_{code}$ 2n+2†
Operand 1: External Operand 2: SARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
Operand 1: DARAM/ ROM Operand 2: External	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
Operand 1: SARAM Operand 2: External	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
Operand 1: External Operand 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	Data Memory 302h	23h
Program Memory FF00h	4h	Program Memory FF00h	4h
TREG	45h	TREG	23h
PREG	458972h	PREG	08Ch
ACC (C = X)	723EC41h	ACC (C = 0)	76975B3h

Example 2

```
MAC     0FF00h,*,AR5   ;(PM = 0, CNF = 1)
```

	Before Instruction		After Instruction
ARP	4	ARP	5
AR4	302h	AR4	302h
Data Memory 302h	23h	Data Memory 302h	23h
Program Memory FF00h	4h	Program Memory FF00h	4h
TREG	45h	TREG	23h
PREG	458972h	PREG	8Ch
ACC (C = X)	723EC41h	ACC (C = 0)	76975B3h

Syntax

MACD *pma*, *dma*	Direct addressing
MACD *pma*, *ind* [, **AR***n*]	Indirect addressing

Operands

dma:	7 LSBs of the data-memory address
pma:	16-bit program-memory address
n:	Value from 0 to 7 designating the next auxiliary register
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

MACD *pma, dma*

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

MACD *pma, ind* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then . . .
(PC) → MSTACK
pma → PC
(ACC) + shifted (PREG) → ACC
(data-memory address) → TREG
(data-memory address) × (pma) → PREG
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC
(data-memory address) → data-memory address + 1

While (repeat counter) ≠ 0:
  (ACC) + shifted (PREG) → ACC
  (data-memory address) → TREG
  (data-memory address) × (pma) → PREG
  For indirect, modify (current AR) and (ARP) as specified
  (PC) + 1 → PC
  (data-memory address) → data-memory address + 1
  (repeat counter) – 1 → repeat counter

(MSTACK) → PC

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

Description

The MACD instruction:

- Adds the previous product, shifted as defined by the PM status bits, to the accumulator. The carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.
- Loads the TREG with the content of the specified data-memory address.
- Multiplies the data-memory value in the TREG by the contents of the specified program-memory address.
- Copies the contents of the specified data-memory address to the next higher data-memory address.

The data- and program-memory locations on the ’C20x may be any nonreserved, on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, the CNF bit must be set to 1. If MACD addresses one of the memory-mapped registers or external memory as a data-memory location, the effect of the instruction is that of a MAC instruction; the data move will not occur (see the DMOV instruction description).

When the MACD instruction is repeated, the program-memory address contained in the PC is incremented by 1 during each repetition. This makes it possible to access a series of operands in program memory. If you use indirect addressing to specify the data-memory address, a new data-memory address can be accessed during each repetition. If you use the direct addressing mode, the specified data-memory address is a constant; it will not be modified during each repetition.

MACD functions in the same manner as MAC, with the addition of a data move for on-chip RAM blocks. This feature makes MACD useful for applications such as convolution and transversal filtering. When used with RPT, MACD becomes a single-cycle instruction once the RPT pipeline is started.

Words 2

Cycles

Cycles for a Single MACD Instruction

Operand	ROM	DARAM	SARAM	External
Operand 1: DARAM/ ROM Operand 2: DARAM	3	3	3	$3{+}2p_{code}$
Operand 1: SARAM Operand 2: DARAM	3	3	3	$3{+}2p_{code}$
Operand 1: External Operand 2: DARAM	$3{+}p_{op1}$	$3{+}p_{op1}$	$3{+}p_{op1}$	$3{+}p_{op1}{+}2p_{code}$
Operand 1: DARAM/ ROM Operand 2: SARAM	3	3	3	$3{+}2p_{code}$
Operand 1: SARAM Operand 2: SARAM	3	3	3 4† 5‡	$3{+}2p_{code}$ $4{+}2p_{code}$†
Operand 1: External Operand 2: SARAM	$3{+}p_{op1}$	$3{+}p_{op1}$	$3{+}p_{op1}$	$3{+}p_{op1}{+}2p_{code}$
Operand 1: DARAM/ ROM Operand 2: External§	$3{+}d_{op2}$	$3{+}d_{op2}$	$3{+}d_{op2}$	$3{+}d_{op2}{+}2p_{code}$
Operand 1: SARAM Operand 2: External§	$3{+}d_{op2}$	$3{+}d_{op2}$	$3{+}d_{op2}$	$3{+}d_{op2}{+}2p_{code}$
Operand 1: External Operand 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 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 of an MACD Instruction

Operand	ROM	DARAM	SARAM	External
Operand 1: DARAM/ ROM Operand 2: DARAM	n+2	n+2	n+2	$n{+}2{+}2p_{code}$
Operand 1: SARAM Operand 2: DARAM	n+2	n+2	n+2	$n{+}2{+}2p_{code}$

† If operand 2 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.

Cycles for a Repeat (RPT) Execution of an MACD Instruction (Continued)

Operand	ROM	DARAM	SARAM	External
Operand 1: External Operand 2: DARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
Operand 1: DARAM/ ROM Operand 2: SARAM	2n	2n	2n 2n+2†	$2n+2p_{code}$
Operand 1: SARAM Operand 2: SARAM	2n 3n‡	2n 3n‡	2n 2n+2† 3n‡ 3n+2§	$2n+2p_{code}$ 3n‡
Operand 1: External Operand 2: SARAM	$2n+np_{op1}$	$2n+np_{op1}$	$2n+np_{op1}$ $2n+2+np_{op1}$†	$2n+np_{op1}+2p_{code}$
Operand 1: DARAM/ ROM Operand 2: External¶	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
Operand 1: SARAM Operand 2: External¶	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
Operand 1: External Operand 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 operand 2 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

```
MACD   0FF00h,08h     ;(DP = 6: addresses 0300h–037Fh,
                      ;PM = 0: no shift of product,
                      ;CNF = 1: RAM B0 configured to
                      ;program memory).
```

	Before Instruction		After Instruction
Data Memory 308h	23h	Data Memory 308h	23h
Data Memory 309h	18h	Data Memory 309h	23h
Program Memory FF00h	4h	Program Memory FF00h	4h
TREG	45h	TREG	23h
PREG	458972h	PREG	8Ch
ACC	C = X; 723EC41h	ACC	C = 0; 76975B3h

Example 2

```
MACD      0FF00h,*,AR6        ;(PM = 0, CNF = 1)
```

	C	Before Instruction		C	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
TREG		45h	TREG		23h
PREG		458972h	PREG		8Ch
ACC	X	723EC41h	ACC	0	76975B3h

Note: The data move function for MACD can occur only within on-chip data memory RAM blocks.

Syntax

MAR *dma*	Direct addressing
MAR *ind* [, **AR***n*]	Indirect addressing

Operands

n: Value from 0 to 7 designating the next auxiliary register

ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

MAR *dma*

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						

MAR *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Event(s)	*Addressing mode*
Increment PC	Direct
Increment PC Modify (current AR) and (ARP) as specified	Indirect

Status Bits

Affects	*Addressing mode*
None	Direct
ARP and ARB	Indirect

Description

In the direct addressing mode, the MAR instruction acts as a NOP instruction.

In the indirect addressing mode, an auxiliary register value and the ARP value can be modified; however, the memory being referenced is not used. When MAR modifies the ARP value, the old ARP value is copied to the ARB field of ST1. Any operation that MAR performs with indirect addressing can also be performed with any instruction that supports indirect addressing. The ARP can also be loaded by an LST instruction.

The LARP instruction from the ’C25 instruction set is a subset of MAR. For example, MAR *, AR4 performs the same function as LARP 4, which loads the ARP with 4.

For loading an auxiliary register, see the description for the LAR instruction. For storing an auxiliary register value to data memory, see the SAR instruction.

Words 1

Cycles

Cycles for a Single MAR Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an MAR Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example 1

```
MAR      *,AR1     ;Load the 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

MPY *dma*	Direct addressing
MPY *ind* [, **AR***n*]	Indirect addressing
MPY #*k*	Short immediate addressing

Operands

dma:	7 LSBs of the data-memory address
n:	Value from 0 to 7 designating the next auxiliary register
k:	13-bit short immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

MPY *dma*

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						

MPY *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

MPY #*k*

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

Execution

Increment PC, then ...

Event	*Addressing mode*
(TREG) × (data-memory address) → PREG	Direct or indirect
(TREG) × k → PREG	Short immediate

Status Bits

None

Description

The contents of TREG are multiplied by the contents of the addressed data memory location. The result is placed in the product register (PREG). With short immediate addressing, TREG is multiplied by a signed 13-bit constant. The short-immediate value is right justified and sign extended before the multiplication, regardless of SXM.

Words

1

Cycles

Cycles for a Single MPY Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an MPY Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 MPY Instruction (Using Short Immediate Addressing)

ROM	DARAM	SARAM	External
1	1	1	1+p

Example 1

```
MPY        DAT13      ;(DP = 8)
```

	Before Instruction		After Instruction
Data Memory 40Dh	7h	Data Memory 40Dh	7h
TREG	6h	TREG	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
TREG	6h	TREG	6h
PREG	36h	PREG	2Ah

Example 3

```
MPY        #031h
```

	Before Instruction		After Instruction
TREG	2h	TREG	2h
PREG	36h	PREG	62h

Syntax

MPYA *dma* — Direct addressing
MPYA *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

MPYA *dma*

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						

MPYA *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) + shifted (PREG) → ACC
(TREG) × (data-memory address) → PREG

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

Description

The contents of TREG are multiplied by the contents of the addressed data memory location. The result is placed in the product register (PREG). The previous product, shifted as defined by the PM status bits, is also added to the accumulator.

Words

1

Cycles

Cycles for a Single MPYA Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an MPYA Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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	Data Memory 30Dh	7h
TREG	6h	TREG	6h
PREG	36h	PREG	2Ah
ACC	X (C) 54h	ACC	0 (C) 8Ah

Example 2

```
MPYA     *,AR4    ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	3	ARP	4
AR3	30Dh	AR3	30Dh
Data Memory 30Dh	7h	Data Memory 30Dh	7h
TREG	6h	TREG	6h
PREG	36h	PREG	2Ah
ACC	X (C) 54h	ACC	0 (C) 8Ah

Syntax

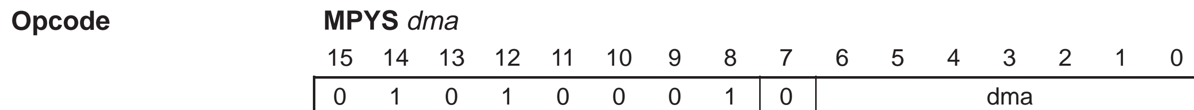
MPYS *dma* — Direct addressing
MPYS *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

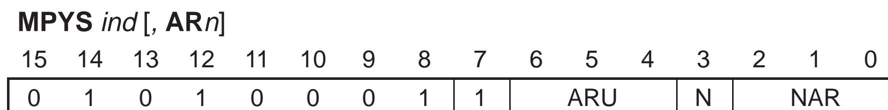
Opcode

MPYS *dma*



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						

MPYS *ind* [, **AR***n*]



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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) – shifted (PREG) → ACC
(TREG) × (data-memory address) → PREG

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

Description

The contents of TREG are multiplied by the contents of the addressed data memory location. The result is placed in the product register (PREG). The previous product, shifted as defined by the PM status bits, is also subtracted from the accumulator, and the result is placed in the accumulator.

Words

1

Cycles

Cycles for a Single MPYS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an MPYS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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)
```

	Before Instruction		After Instruction
Data Memory 30Dh	7h	Data Memory 30Dh	7h
TREG	6h	TREG	6h
PREG	36h	PREG	2Ah
ACC	C: X; 54h	ACC	C: 1; 1Eh

Example 2

```
MPYS     *,AR5     ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	4	ARP	5
AR4	30Dh	AR4	30Dh
Data Memory 30Dh	7h	Data Memory 30Dh	7h
TREG	6h	TREG	6h
PREG	36h	PREG	2Ah
ACC	C: X; 54h	ACC	C: 1; 1Eh

Syntax

MPYU *dma*	Direct addressing
MPYU *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

MPYU *dma*

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						

MPYU *ind* [,**AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
Unsigned (TREG) × unsigned (data-memory address) → PREG

Status Bits

None

This instruction is not affected by SXM.

Description

The unsigned contents of TREG are multiplied by the unsigned contents of the addressed data-memory location. The result is placed in the product register (PREG). The multiplier acts as a signed 17 × 17-bit multiplier for this instruction, with the MSB of both operands forced to 0.

When another instruction passes the resulting PREG value to data memory or to the CALU, the value passes first through the product shifter at the output of the PREG. This shifter always invokes sign extension on the PREG value when PM = 3 (right-shift-by-6 mode). Therefore, this shift mode should not be used if unsigned products are desired.

The MPYU instruction is particularly useful for computing multiple-precision products, such as when multiplying two 32-bit numbers to yield a 64-bit product.

Words

1

Cycles

Cycles for a Single MPYU Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an MPYU Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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     16        ;(DP = 4: addresses 0200h–027Fh)
```

	Before Instruction		After Instruction
Data Memory 210h	0FFFFh	Data Memory 210h	0FFFFh
TREG	0FFFFh	TREG	0FFFFh
PREG	1h	PREG	0FFFE0001h

Example 2

```
MPYU     *,AR6
```

	Before Instruction		After Instruction
ARP	5	ARP	6
AR5	210h	AR5	210h
Data Memory 210h	0FFFFh	Data Memory 210h	0FFFFh
TREG	0FFFFh	TREG	0FFFFh
PREG	1h	PREG	0FFFE0001h

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 Increment PC, then ...
(ACC) × –1 → ACC

Status Bits

Affected by	*Affects*
OVM	C and OV

Description The content of the accumulator is replaced with its arithmetic complement (2s complement). The OV bit is set when taking the NEG of 8000 0000h. If OVM = 1, the accumulator content is replaced with 7FFF FFFFh. If OVM = 0, the result is 8000 0000h. The carry bit (C) is cleared to 0 by this instruction for all nonzero values of the accumulator, and is set to 1 if the accumulator equals zero.

Words 1

Cycles

Cycles for a Single NEG Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an NEG Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example 1

```
NEG        ;(OVM = X) Convert -3544 to +3544
```

	Before Instruction		After Instruction
ACC	0FFFFF228h	ACC	0DD8h
C	X	C	0
OV	X	OV	X

Example 2

```
NEG        ;(OVM = 0)
```

	Before Instruction		After Instruction
ACC	080000000h	ACC	080000000h
C	X	C	0
OV	X	OV	1

Example 3

```
NEG        ;(OVM = 1)
```

	Before Instruction			After Instruction	
ACC	X	080000000h	ACC	0	7FFFFFFFh
	C			C	
	X			1	
	OV			OV	

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

Affects
INTM

This instruction is not affected by INTM.

Description

The NMI instruction forces the program counter to the nonmaskable interrupt vector located at 24h. This instruction has the same effect as the hardware nonmaskable interrupt $\overline{\text{NMI}}$.

Words 1

Cycles

Cycles for a Single NMI Instruction

ROM	DARAM	SARAM	External
4	4	4	4+3p†

† The ’C20x performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken, these two instruction words are discarded.

Example

```
NMI   ;PC + 1 is pushed onto the stack, and then
      ;control is passed to program memory location
      ;24h.
```

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 Increment PC

Status Bits None

Description No operation is performed. The NOP instruction affects only the PC. The NOP instruction is useful for creating pipeline and execution delays.

Words 1

Cycles

Cycles for a Single NOP Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an NOP Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
NOP         ;No operation is performed.
```

Syntax **NORM** *ind* Indirect addressing

Operands ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

NORM *ind*

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution Increment PC, then ...

If (ACC) = 0:
  Then TC → 1;
Else, if (ACC(31)) XOR (ACC(30)) = 0:
    Then TC → 0,
      (ACC) × 2 → ACC
      Modify (current AR) as specified;
    Else TC → 1.

Status Bits *Affects*
TC

Description The NORM instruction normalizes a signed number that is contained in the accumulator. Normalizing a fixed-point number separates it into a mantissa and an exponent by finding the magnitude of the sign-extended number. An exclusive-OR operation is performed on accumulator bits 31 and 30 to determine if bit 30 is part of the magnitude or part of the sign extension. If they are the same, they are both sign bits, and the accumulator is left shifted to eliminate the extra sign bit.

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 accumulator. 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. NORM functions on both positive and negative 2s-complement numbers.

Notes:

For the NORM instruction, the auxiliary register operations are executed during the fourth phase of the pipeline, the execution phase. For other instructions, the auxiliary register operations take place in the second phase of the pipeline, in the decode phase. Therefore:

1) **The auxiliary register values should not be modified by the two instruction words following NORM.** If the auxiliary register used in the NORM instruction is to be affected by either of the next two instruction words, the auxiliary register value will be modified by the other instructions *before* it is modified by the NORM instruction.

2) **The value in the auxiliary register pointer (ARP) should not be modified by the two instruction words following NORM.** If either of the next two instruction words specify a change in the ARP value, the ARP value will be changed *before* NORM is executed; the ARP will not be pointing at the correct auxiliary register when NORM is executed.

Words 1

Cycles

Cycles for a Single NORM Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of a NORM Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example 1

```
NORM     *+
```

	Before Instruction		After Instruction
ARP	2	ARP	2
AR2	00h	AR2	01h
ACC	X (C) 0FFFFF001h	ACC	X (C) 0FFFE002h
TC	X	TC	0

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 used in Example 2 normalizes a 32-bit number and yields a 5-bit exponent magnitude. The method used 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.

Syntax

OR *dma*	Direct addressing
OR *ind* [, **AR***n*]	Indirect addressing
OR #*lk* [, *shift*]	Long immediate addressing
OR #*lk,* **16**	Long immediate with left shift of 16

Operands

dma:	7 LSBs of the data-memory address
shift:	Left shift value from 0 to 15 (defaults to 0)
n:	Value from 0 to 7 designating the next auxiliary register
lk:	16-bit long immediate value
ind:	Select one of the following seven options: * *+ *– *0+ *0– *BR0+ *BR0–

Opcode

OR *dma*

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

OR *ind* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

OR #*lk* [, *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	0	shift			
lk															

OR #*lk* [, **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	0
lk															

Execution

Increment PC, then ...

Event(s)	*Addressing mode*
(ACC(15:0)) OR (data-memory address) $\rightarrow$ ACC(15:0) (ACC(31:16)) $\rightarrow$ ACC(31:16)	Direct or indirect
(ACC) OR lk $\times$ 2^{shift} $\rightarrow$ ACC	Long immediate
(ACC) OR lk $\times$ 2^{16} $\rightarrow$ ACC	Long immediate with left shift of 16

Status Bits

None
This instruction is not affected by SXM.

Description

An OR operation is performed on the contents of the accumulator and the contents of the addressed data-memory location or a long-immediate value. The long-immediate value may be shifted before the OR operation. The result remains in the accumulator. All bit positions unoccupied by the data operand are zero filled, regardless of the value of the SXM status bit. Thus, the high word of the accumulator is unaffected by this instruction if direct or indirect addressing is used, or if immediate addressing is used with a shift of 0. Zeros are shifted into the least significant bits of the operand if immediate addressing is used with a nonzero shift count.

Words

Words	*Addressing mode*
1	Direct or indirect
2	Long immediate

Cycles

Cycles for a Single OR Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an OR Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 OR Instruction (Using Long Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+2p

Example 1

```
OR DAT8    ;(DP = 8)
```

		Before Instruction			After Instruction
Data Memory 408h		0F000h	Data Memory 408h		0F000h
ACC	X (C)	100002h	ACC	X (C)	10F002h

Example 2

```
OR *,AR0
```

		Before Instruction			After Instruction
ARP		1	ARP		0
AR1		300h	AR1		300h
Data Memory 300h		1111h	Data Memory 300h		1111h
ACC	X (C)	222h	ACC	X (C)	1333h

Example 3

```
OR #08111h,8
```

		Before Instruction			After Instruction
ACC	X (C)	0FF0000h	ACC	X (C)	0FF1100h

Syntax

OUT *dma*, *PA*	Direct addressing
OUT *ind*, *PA* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
PA: 16-bit I/O address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
*   *+   *–   *0+   *0–   *BR0+   *BR0–

Opcode

OUT *dma*, *PA*

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

OUT *ind*, *PA* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
PA → address bus A15–A0
(data-memory address) → data bus D15–D0
(data-memory address) → PA

Status Bits

None

Description

The OUT instruction writes a 16-bit value from a data-memory location to the specified I/O location. The $\overline{\text{IS}}$ line goes low to indicate an I/O access. The $\overline{\text{STRB}}$, R/$\overline{\text{W}}$, and READY timings are the same as for an external data-memory write.

RPT can be used with the OUT instruction to write consecutive words from data memory to I/O space.

Words

2

Cycles

Cycles for a Single OUT Instruction

Operand	Program			
	ROM	DARAM	SARAM	External
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 operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution of an OUT Instruction

Operand	Program			
	ROM	DARAM	SARAM	External
Destination: DARAM	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+3+nio_{dst}+2p_{code}$
Destination: SARAM	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$ $3n+1+nio_{dst}$†	$3n+3+nio_{dst}+2p_{code}$
Destination: 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 operand and the code are in the same SARAM block

Example 1

```
OUT   DAT0,100h     ;(DP = 4) Write data word stored in
                    ;data memory location 200h to
                    ;peripheral at I/O port address
                    ;100h.
```

Example 2

```
OUT   *,100h        ;Write data word referenced by
                    ;current auxiliary register to
                    ;peripheral at I/O port address
                    ;100h.
```

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 Increment PC, then ...
shifted (PREG) → ACC

Status Bits *Affected by*
PM

Description The content of PREG, shifted as specified by the PM status bits, is loaded into the accumulator.

Words 1

Cycles

Cycles for a Single PAC Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of a PAC Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
PAC        ;(PM = 0: no shift of product)
```

	C	Before Instruction		C	After Instruction
PREG		144h	PREG		144h
ACC	X	23h	ACC	X	144h

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

Increment PC, then ...
(TOS) → ACC(15:0)
0 → ACC(31:16)
Pop stack one level

Status Bits None

Description

The content of the top of the stack (TOS) is copied to the low accumulator, and then the stack values move up one level. The upper half of the accumulator is set to all zeros.

The hardware stack functions as a last-in, first-out stack 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 will have the same value. Because each stack value is copied, if more than seven stack pops (using the POP, POPD, RETC, or RET instructions) occur before any pushes occur, all levels of the stack will contain the same value. No provision exists to check stack underflow.

Words 1

Cycles

Cycles for a Single POP Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of a POP Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example `POP`

		Before Instruction			After Instruction
ACC	X	82h	ACC	X	45h
	C			C	
Stack		45h	Stack		16h
		16h			7h
		7h			33h
		33h			42h
		42h			56h
		56h			37h
		37h			61h
		61h			61h

Syntax

POPD *dma* — Direct addressing
POPD *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

POPD *dma*

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						

POPD *ind* [,**AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(TOS) → data-memory address
Pop stack one level

Status Bits

None

Description

The value from the top of the stack is transferred into the data-memory location specified by the instruction. In the lower seven locations of the stack, the values are copied up one level. The stack operation is explained in the description for the POP instruction. No provision exists to check stack underflow.

Words

1

Cycles

Cycles for a Single POPD Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of a POPD Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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		0AAh
	0AAh		0AAh

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		0AAh
	0AAh		0AAh

Syntax

PSHD *dma* — Direct addressing
PSHD *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

PSHD *dma*

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						

PSHD *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → TOS
Push all stack locations down one level

Status Bits

None

Description

The value from the data-memory location specified by the instruction is transferred to the top of the stack. In the lower seven locations of the stack, the values are also copied one level down, as explained in the description for the PUSH instruction. The value in the lowest stack location is lost.

Words

1

Cycles

Cycles for a Single PSHD Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of a PSHD Instruction

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

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

Example 1

```
PSHD     127       ;(DP = 3: addresses 0180–01FFh)
```

	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 Increment PC, then...
Push all stack locations down one level
ACC(15:0) → TOS

Status Bits None

Description The stack values move down one level. Then, the content of the lower half of the accumulator is copied onto the top of the hardware stack.

The hardware stack operates as a last-in, first-out stack with eight locations. If more than eight pushes (due to a CALA, CALL, CC, PSHD, PUSH, TRAP, INTR, or NMI instruction) occur before a pop, the first data values written are lost with each succeeding push.

Words 1

Cycles

Cycles for a Single PUSH Instruction

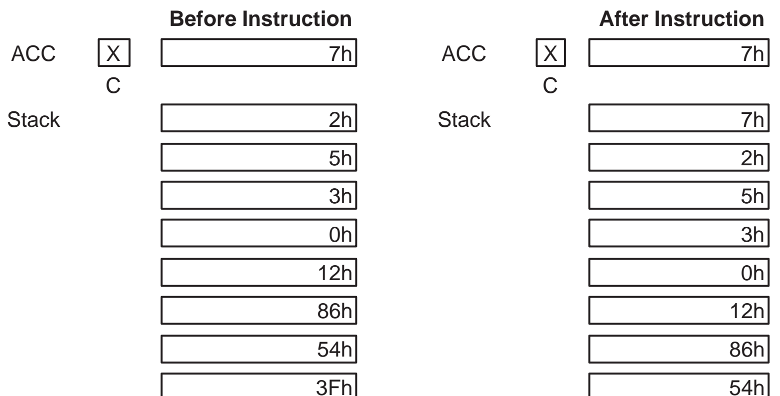
ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of a PUSH Instruction

ROM	DARAM	SARAM	External
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

Description The contents of the top stack register are copied into the program counter. The remaining stack values are then copied up one level. RET concludes subroutines and interrupt service routines to return program control to the calling or interrupted program sequence.

Words 1

Cycles

Cycles for a Single RET Instruction

ROM	DARAM	SARAM	External
4	4	4	4+3p

Note: When this instruction reaches the execute phase of the pipeline, two additional instruction words have entered the pipeline. When the 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* 1 [, *cond* 2] [,...]

Operands

cond	*Condition*
EQ	ACC = 0
NEQ	ACC ≠ 0
LT	ACC < 0
LEQ	ACC ≤ 0
GT	ACC > 0
GEQ	ACC ≥ 0
NC	C = 0
C	C =1
NOV	OV = 0
OV	OV = 1
BIO	$\overline{\text{BIO}}$ low
NTC	TC = 0
TC	TC = 1
UNC	Unconditionally

‡

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			

Note: The TP and ZLVC fields are defined on pages 7-3 and 7-4.

Execution If *cond* 1 AND *cond* 2 AND ...
(TOS) → PC
Pop stack one level
Else, continue

Status Bits None

Description If the specified condition or conditions are met, a standard return is executed (see the description for the RET instruction). Note that not all combinations of conditions are meaningful. For example, testing for LT and GT is contradictory. In addition, testing $\overline{\text{BIO}}$ is mutually exclusive to testing TC.

Words 1

Cycles

Cycles for a Single RETC Instruction

Condition	ROM	DARAM	SARAM	External
True	4	4	4	4+4p
False	2	2	2	2+2p

Note: The processor performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken, these two instruction words are discarded.

Example

```
RETC      GEQ,NOV   ;A return is executed if the
                    ;accumulator content is positive
                    ;or zero and if the OV (overflow)
                    ;-bit is zero.
```

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 Increment PC, then ...
C → ACC(0)
(ACC(31)) → C
(ACC(30:0)) → ACC(31:1)

Status Bits *Affects*
C

This instruction is not affected by SXM.

Description The ROL instruction rotates the accumulator left one bit. The value of the carry bit is shifted into the LSB, then the MSB is shifted into the carry bit.

Words 1

Cycles

Cycles for a Single ROL Instruction

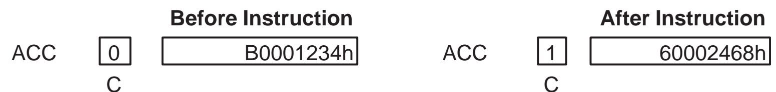
ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an ROL Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
ROL
```



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
Increment PC, then ...
C → ACC(31)
(ACC(0)) → C
(ACC(31:1)) → ACC(30:0)

Status Bits
Affects
C

This instruction is not affected by SXM.

Description The ROR instruction rotates the accumulator right one bit. The value of the carry bit is shifted into the MSB of the accumulator, then the LSB of the accumulator is shifted into the carry bit.

Words 1

Cycles

Cycles for a Single ROR Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an ROR Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
ROR
```

Syntax

RPT *dma*	Direct addressing
RPT *ind* [, **AR***n*]	Indirect addressing
RPT #*k*	Short immediate

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
k: 8-bit short immediate value
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

RPT *dma*

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						

RPT *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

RPT #*k*

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

Execution

Increment PC, then ...

Event	*Addressing mode*
(data-memory address) → RPTC	Direct or indirect
k → RPTC	Short immediate

Status Bits

None

Description

The repeat counter (RPTC) is loaded with the content of the addressed data-memory location if direct or indirect addressing is used; it is loaded with an 8-bit immediate value if short immediate addressing is used. The instruction following the RPT is repeated *n* times, where *n* is the initial value of the RPTC plus 1. Since the RPTC cannot be saved during a context switch, repeat loops are regarded as multicycle instructions and are not interruptible. The RPTC is cleared to 0 on a device reset.

RPT is especially useful for block moves, multiply/accumulates, and normalization. The repeat instruction itself is not repeatable.

Words

1

Cycles

Cycles for a Single RPT Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 RPT Instruction (Using Short Immediate Addressing)

ROM	DARAM	SARAM	External
1	1	1	1+p

Example 1

```
RPT   DAT127     ;(DP = 31: addresses 0F80h–0FFFh)
                 ;Repeat next instruction 13 times.
```

	Before Instruction		After Instruction
Data Memory 0FFFh	0Ch	Data Memory 0FFFh	0Ch
RPTC	0h	RPTC	0Ch

Example 2

```
RPT   *,AR1      ;Repeat next instruction 4096 times.
```

	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 two times.
```

	Before Instruction		After Instruction
RPTC	0h	RPTC	1h

Syntax

SACH *dma* [, *shift2*]	Direct addressing
SACH *ind* [, *shift2* [, **AR***n*]]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
shift2: Left shift value from 0 to 7 (defaults to 0)
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SACH *dma* [, *shift2*]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	shift2			0	dma						

SACH *ind* [, *shift2*[, **AR***n*]]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	shift2			1	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
16 MSBs of ((ACC) $\times$ 2^{shift2}) $\rightarrow$ data-memory address

Status Bits

This instruction is not affected by SXM

Description

The SACH instruction copies the entire accumulator into the output shifter, where it left shifts the entire 32-bit number from 0 to 7 bits. It then copies the upper 16 bits of the shifted value into data memory. During the shift, the low-order bits are filled with zeros, and the high-order bits are lost. The accumulator itself remains unaffected.

Words

1

Cycles

Cycles for a Single SACH Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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

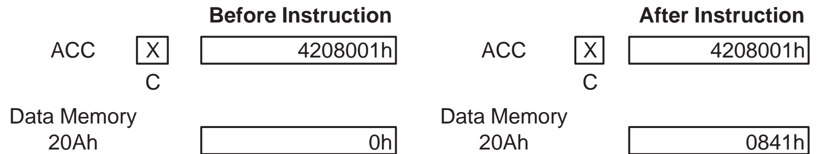
Cycles for a Repeat (RPT) Execution of an SACH Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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

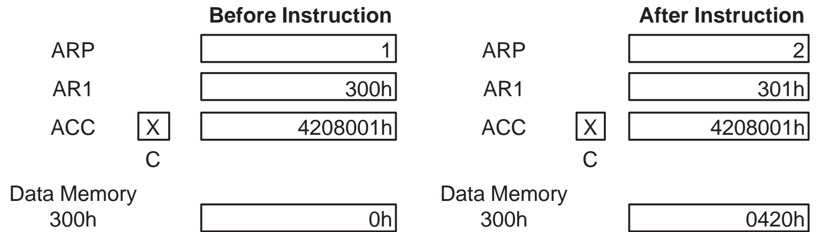
Example 1

```
SACH      DAT10,1   ;(DP = 4: addresses 0200h–027Fh,
                    ;left shift of 1)
```



Example 2

```
SACH      *+,0,AR2  ;(No shift)
```



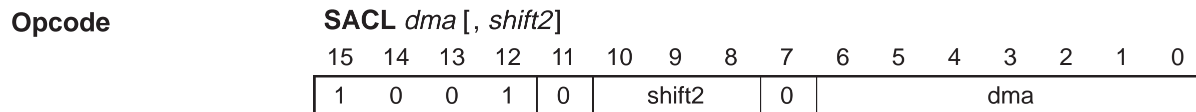
Syntax

SACL *dma* [, *shift2*] — Direct addressing
SACL ind [, *shift2* [, **AR***n*]] — Indirect addressing

Operands

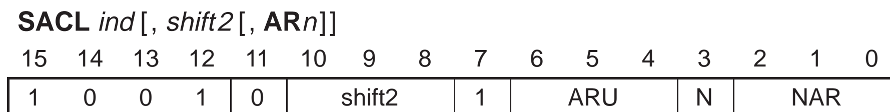
dma: 7 LSBs of the data-memory address
shift2: Left shift value from 0 to 7 (defaults to 0)
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode



SACL *dma* [, *shift2*]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	shift2			0	dma						



SACL *ind* [, *shift2* [, **AR***n*]]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	shift2			1	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
16 LSBs of ((ACC) $\times$ 2^{shift2}) $\rightarrow$ data-memory address

Status Bits

This instruction is not affected by SXM.

Description

The SACL instruction copies the entire accumulator into the output shifter, where it left shifts the entire 32-bit number from 0 to 7 bits. It then copies the lower 16 bits of the shifted value into data memory. During the shift, the low-order bits are filled with zeros, and the high-order bits are lost. The accumulator itself remains unaffected.

Words

1

Cycles

Cycles for a Single SACL Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SACL Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0200h–027Fh,
                    ;left shift of 1)
```

	Before Instruction			After Instruction	
ACC	X (C)	7C63 8421	ACC	X (C)	7C63 8421h
Data Memory 20Bh		05h	Data Memory 20Bh		0842h

Example 2

```
SACL      *,0,AR7   ;(No shift)
```

	Before Instruction			After Instruction	
ARP		6	ARP		7
AR6		300h	AR6		300h
ACC	X (C)	00FF 8421h	ACC	X (C)	00FF 8421h
Data Memory 300h		05h	Data Memory 300h		8421h

Syntax

SAR AR*x*, *dma*	Direct addressing
SAR AR*x,* ind [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
x: Value from 0 to 7 designating the auxiliary register value to be stored
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
`*  *+  *–  *0+  *0–  *BR0+  *BR0–`

Opcode

SAR AR*x*, *dma*

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

SAR AR*x*, *ind* [*,* **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ARx) → data-memory address

Status Bits

None

Description

The content of the designated auxiliary register (ARx) is stored in the specified data-memory location. When the content of the designated auxiliary register is also modified by the instruction (in indirect addressing mode), SAR copies the auxiliary register value to data memory before it increments or decrements the contents of the auxiliary register.

Words

1

Cycles

Cycles for a Single SAR Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SAR Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0300h–037Fh)
```

	Before Instruction		After Instruction
AR0	37h	AR0	37h
Data Memory 31Eh	18h	Data Memory 31Eh	37h

Example 2

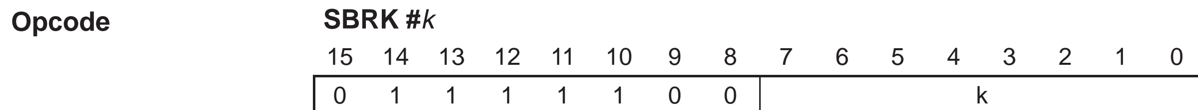
```
SAR     AR0,*+
```

	Before Instruction		After Instruction
ARP	0	ARP	0
AR0	401h	AR0	402h
Data Memory 401h	0h	Data Memory 401h	401h

Syntax **SBRK #*k*** Short immediate addressing

Operands k: 8-bit positive short immediate value

Opcode



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

Execution Increment PC, then ...
(current AR) – k → current AR

Note that k is an 8-bit positive constant.

Status Bits None

Description The 8-bit immediate value is subtracted, right justified, from the content of the current auxiliary register (the one pointed to by the ARP) and the result replaces the contents of the auxiliary register. The subtraction 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 auxiliary registers are unsigned.

Words 1

Cycles

Cycles for a Single SBRK Instruction

ROM	DARAM	SARAM	External
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: Select one of the following control bits:

C	Carry bit of status register ST1
CNF	RAM configuration control bit of status register ST1
INTM	Interrupt mode bit of status register ST0
OVM	Overflow mode bit of status register ST0
SXM	Sign-extension mode bit of status register ST1
TC	Test/control flag bit of status register ST1
XF	XF pin status bit of status register ST1

Opcode

SETC C

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 CNF

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

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

SETC OVM

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

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 TC

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 XF

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

Execution Increment PC, then ...
1 → control bit

Status Bits None

Description The specified control bit is set to 1. Note that LST may also be used to load ST0 and ST1. See section 3.5, *Status and Control Registers*, on page 3-15 for more information on each control bit.

Words 1

Cycles

Cycles for a Single SETC Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an SETC Instruction

ROM	DARAM	SARAM	External
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

Increment PC, then ...
(ACC(31)) → C
(ACC(30:0)) → ACC(31:1)
0 → ACC(0)

Status Bits

Affects
C

This instruction is not affected by SXM.

Description

The SFL instruction shifts the entire accumulator left one bit. The least significant bit is filled with a 0, and the most significant bit is shifted into the carry bit (C). SFL, unlike SFR, is unaffected by SXM.

Words 1

Cycles

Cycles for a Single SFL Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an SFL Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
SFL
```

	Before Instruction			After Instruction	
	C			C	
ACC	X	B0001234h	ACC	1	60002468h

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

Increment PC, then ...
If SXM = 0
  Then 0 → ACC(31).
If SXM = 1
  Then (ACC(31)) → ACC(31)

(ACC(31:1)) → ACC(30:0)
(ACC(0)) → C

Status Bits

Affected by	*Affects*
SXM	C

Description The SFR instruction shifts the accumulator right one bit.

- If SXM = 1, the instruction produces an arithmetic right shift. The sign bit (MSB) is unchanged and is also copied into bit 30. Bit 0 is shifted into the carry bit (C).
- If SXM = 0, the instruction produces a logic right shift. All of the accumulator bits are shifted right by one bit. The least significant bit is shifted into the carry bit, and the most significant bit is filled with a 0.

Words 1

Cycles

Cycles for a Single SFR Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an SFR Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example 1

```
SFR         ;(SXM = 0: no sign extension)
```

	C	Before Instruction		C	After Instruction
ACC	X	B0001234h	ACC	0	5800091Ah

Example 2

```
SFR         ;(SXM = 1: sign extend)
```

	C	Before Instruction		C	After Instruction
ACC	X	B0001234h	ACC	0	D800091Ah

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 Increment PC, then ...
(ACC) – shifted (PREG) → ACC

Status Bits

Affected by	*Affects*
PM and OVM	C and OV

This instruction is not affected by SXM.

Description The content of PREG, shifted as defined by the PM status bits, is subtracted from the content of the accumulator. The result is stored in the accumulator. SPAC is not affected by SXM, and the PREG value is always sign extended.

The function of the SPAC instruction is a subtask of the LTS, MPYS, and SQRS instructions.

Words 1

Cycles

Cycles for a Single SPAC Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Repeat (RPT) Execution of an SPAC Instruction

ROM	DARAM	SARAM	External
n	n	n	n+p

Example

```
SPAC        ;(PM = 0)
```

	C	Before Instruction		C	After Instruction
PREG		10000000h	PREG		10000000h
ACC	X	70000000h	ACC	1	60000000h

Syntax

SPH *dma* — Direct addressing
SPH *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SPH *dma*

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						

SPH *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
16 MSBs of shifted (PREG) → data-memory address

Status Bits

Affected by
PM

Description

The 16 high-order bits of the PREG, shifted as specified by the PM bits, are stored in data memory. First, the 32-bit PREG value is copied into the product shifter, where it is shifted as specified by the PM bits. If the right-shift-by-6 mode is selected, the high-order bits are sign extended and the low-order bits are lost. If a left shift is selected, the high-order bits are lost and the low-order bits are zero filled. If PM = 00, no shift occurs. Then the 16 MSBs of the shifted value are stored in data memory. Neither the PREG value nor the accumulator value is modified by this instruction.

Words

1

Cycles

Cycles for a Single SPH Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SPH Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0200h–027Fh,
                  ;PM = 0: no shift)
```

	Before Instruction		After Instruction
PREG	FE079844h	PREG	FE079844h
Data Memory 203h	4567h	Data Memory 203h	FE07h

Example 2

```
SPH     *,AR7     ;(PM = 2: left shift of four)
```

	Before Instruction		After Instruction
ARP	6	ARP	7
AR6	203h	AR6	203h
PREG	FE079844h	PREG	FE079844h
Data Memory 203h	4567h	Data Memory 203h	E079h

Syntax

SPL *dma* — Direct addressing
SPL *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SPL *dma*

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						

SPL *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
16 LSBs of shifted (PREG) → data-memory address

Status Bits

<u>*Affected by*</u>
PM

Description

The 16 low-order bits of the PREG, shifted as specified by the PM bits, are stored in data memory. First, the 32-bit PREG value is copied into the product shifter, where it is shifted as specified by the PM bits. If the right-shift-by-6 mode is selected, the high-order bits are sign extended and the low-order bits are lost. If a left shift is selected, the high-order bits are lost and the low-order bits are zero filled. If PM = 00, no shift occurs. Then the 16 LSBs of the shifted value are stored in data memory. Neither the PREG value nor the accumulator value is modified by this instruction.

Words

1

Cycles

Cycles for a Single SPL Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SPL Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 = 4: addresses 0200h–027Fh,
                 ;PM = 2: left shift of four)
```

	Before Instruction		After Instruction
PREG	0FE079844h	PREG	0FE079844h
Data Memory 205h	4567h	Data Memory 205h	08440h

Example 2

```
SPL     *,AR3    ;(PM = 0: no shift)
```

	Before Instruction		After Instruction
ARP	2	ARP	3
AR2	205h	AR2	205h
PREG	0FE079844h	PREG	0FE079844h
Data Memory 205h	4567h	Data Memory 205h	09844h

Syntax

SPLK #*lk*, *dma*	Direct addressing
SPLK #*lk*, *ind* [, **AR***n*]	Indirect addressing

Operands

dma:	7 LSBs of the data-memory address
n:	Value from 0 to 7 designating the next auxiliary register
lk:	16-bit long immediate value
ind:	Select one of the following seven options: * *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SPLK *#lk, dma*

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

SPLK *#lk, ind* [, **AR***n*]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
lk → data-memory address

Status Bits

None

Description

The SPLK instruction allows a full 16-bit pattern to be written into any data memory location.

Words

2

Cycles

Cycles for a Single SPLK Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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)
```

	Before Instruction		After Instruction
Data Memory 303h	FE07h	Data Memory 303h	7FFFh

Example 2

```
SPLK    #1111h,*+,AR4
```

	Before Instruction		After Instruction
ARP	0	ARP	4
AR0	300h	AR0	301h
Data Memory 300h	07h	Data Memory 300h	1111h

Syntax **SPM** *constant*

Operands constant: Value from 0 to 3 that determines the product shift mode

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	constant	

Execution Increment PC, then ...
constant → product shift mode (PM) bits

Status Bits *Affects*
PM

This instruction is not affected by SXM.

Description The two LSBs of the instruction word are copied into the product shift mode (PM) bits of status register ST1 (bits 1 and 0 of ST1). The PM bits control the mode of the shifter at the output of the PREG. This shifter can shift the PREG output either one or four bits to the left or six bits to the right. The possible PM bit combinations and their meanings are shown in Table 7–8. When an instruction accesses the PREG value, the value first passes through the shifter, where it is shifted by the specified amount.

Table 7–8. Product Shift Modes

PM Field	Specified Product Shift
00	No shift of PREG output
01	PREG output to be left shifted 1 place
10	PREG output to be left shifted 4 places
11	PREG output to be right shifted 6 places and sign extended

The left shifts allow the product to be justified for fractional arithmetic. The right-shift-by-six mode allows up to 128 multiply accumulate processes without the possibility of overflow occurring. PM may also be loaded by an LST #1 instruction.

Words 1

Cycles

Cycles for a Single SPM Instruction

ROM	DARAM	SARAM	External
1	1	1	1+p

Example

```
SPM   3     ;Product register shift mode 3 (PM = 11)
            ;is selected causing all subsequent
            ;transfers from the product register (PREG)
            ;to be shifted to the right six places.
```

Syntax

SQRA *dma* — Direct addressing
SQRA *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SQRA *dma*

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						

SQRA *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) + shifted (PREG) → ACC
(data-memory address) → TREG
(TREG) × (data-memory address) → PREG

Status Bits

Affected by	Affects
OVM and PM	OV and C

Description

The content of the PREG, shifted as defined by the PM status bits, is added to the accumulator. Then the addressed data-memory value is loaded into the TREG, squared, and stored in the PREG.

Words

1

Cycles

Cycles for a Single SQRA Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SQRA Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0300h–037Fh,
                    ;PM = 0: no shift of product)
```

	Before Instruction		After Instruction
Data Memory 31Eh	0Fh	Data Memory 31Eh	0Fh
TREG	3h	TREG	0Fh
PREG	12Ch	PREG	0E1h
ACC (C = X)	1F4h	ACC (C = 0)	320h

Example 2

```
SQRA      *,AR4     ;(PM = 0)
```

	Before Instruction		After Instruction
ARP	3	ARP	4
AR3	31Eh	AR3	31Eh
Data Memory 31Eh	0Fh	Data Memory 31Eh	0Fh
TREG	3h	TREG	0Fh
PREG	12Ch	PREG	0E1h
ACC (C = X)	1F4h	ACC (C = 0)	320h

Syntax

SQRS *dma* — Direct addressing
SQRS *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SQRS *dma*

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						

SQRS *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) – shifted (PREG) → ACC
(data-memory address) → TREG
(TREG) × (data-memory address) → PREG

Status Bits

Affected by	*Affects*
OVM and PM	OV and C

Description

The content of the PREG, shifted as defined by the PM status bits, is subtracted from the accumulator. Then the addressed data-memory value is loaded into the TREG, squared, and stored in the PREG.

Words

1

Cycles

Cycles for a Single SQRS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SQRS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0300h–037Fh,
                    ;PM = 0: no shift of product)
```

	Before Instruction		After Instruction
Data Memory 309h	08h	Data Memory 309h	08h
TREG	1124h	TREG	08h
PREG	190h	PREG	40h
ACC (C = X)	1450h	ACC (C = 1)	12C0h

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
TREG	1124h	TREG	08h
PREG	190h	PREG	40h
ACC (C = X)	1450h	ACC (C = 1)	12C0h

Syntax

SST #*m*, *dma*	Direct addressing
SST #*m*, *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
m: Select one of the following:
- 0 Indicates that ST0 will be stored
- 1 Indicates that ST1 will be stored

ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SST #0, *dma*

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						

SST #0, *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

SST #1, *dma*

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						

SST #1, *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(status register STm) → data-memory address

Status Bits

None

Description

Status register ST0 or ST1 (whichever is specified) is stored in data memory.

In direct addressing mode, the specified status register is always stored in data page 0, regardless of the value of the data page pointer (DP) in ST0. Although the processor automatically accesses page 0, the DP is not physically modified; this allows the DP value to be stored unchanged when ST0 is stored.

In indirect addressing mode, the storage address is obtained from the auxiliary register selected; thus, the specified status register contents can be stored to an address on any page in data memory.

Status registers ST0 and ST1 are defined in section 3.5, *Status Registers ST0 and ST1*, on page 3-15.

Words 1

Cycles

Cycles for a Single SST Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SST Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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     #0,96     ;Direct addressing: data page 0
                  ;accessed automatically
```

	Before Instruction		After Instruction
ST0	0A408h	ST0	0A408h
Data Memory 60h	0Ah	Data Memory 60h	0A408h

Example 2

```
SST     #1,*,AR7  ;Indirect addressing
```

	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

SUB *dma* [, *shift*]	Direct addressing
SUB *dma*,**16**	Direct with left shift of 16
SUB *ind* [,*shift* [, **AR***n*]]	Indirect addressing
SUB *ind*,**16**[, **AR***n*]	Indirect with left shift of 16
SUB #*k*	Short immediate
SUB #*lk* [,*shift*]	Long immediate

Operands

dma:	7 LSBs of the data-memory address
shift:	Left shift value from 0 to 15 (defaults to 0)
n:	Value from 0 to 7 designating the next auxiliary register
k:	8-bit short immediate value
lk:	16-bit long immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SUB *dma* [,*shift*]

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

SUB *dma,* **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						

SUB *ind* [, *shift* [, **AR***n*]]

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

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

SUB *ind*,**16** [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

SUB #*k*

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

SUB #*lk* [, *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	shift			
lk															

Execution Increment PC, then ...

Event	*Addressing mode*
(ACC) – ((data-memory address) $\times$ 2^{shift}) $\rightarrow$ ACC	Direct or indirect
(ACC) – ((data-memory address) $\times$ 2^{16}) $\rightarrow$ ACC	Direct or indirect (shift of 16)
(ACC) – k $\rightarrow$ ACC	Short immediate
(ACC) – lk $\times$ 2^{shift} $\rightarrow$ ACC	Long immediate

Status Bits

Affected by	*Affects*	*Addressing mode*
OVM and SXM	OV and C	Direct or indirect
OVM	OV and C	Short immediate
OVM and SXM	OV and C	Long immediate

Description In direct, indirect, and long immediate addressing, the content of the addressed data-memory location or a 16-bit constant are left shifted and subtracted from the accumulator. During shifting, low-order bits are zero filled. High-order bits are sign extended if SXM = 1 and zero filled if SXM = 0. The result is then stored in the accumulator.

If short immediate addressing is used, an 8-bit positive constant is subtracted from the accumulator. In this case, no shift value may be specified, the subtraction is unaffected by SXM, and the instruction is not repeatable.

Normally, the carry bit is cleared (C = 0) if the result of the subtraction generates a borrow; it is set (C = 1) if it does not generate a borrow. However, if a 16-bit shift is specified with the subtraction, the instruction will clear the carry bit if a borrow is generated but will not affect the carry bit otherwise.

Words

Words	*Addressing mode*
1	Direct, indirect or short immediate
2	Long immediate

Cycles

Cycles for a Single SUB Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SUB Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 SUB Instruction (Using Short Immediate Addressing)

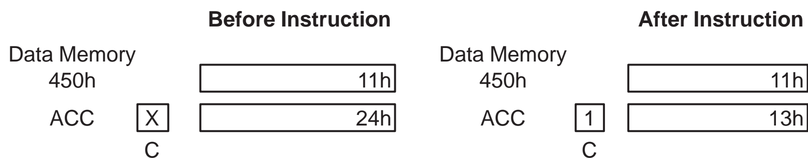
ROM	DARAM	SARAM	External
1	1	1	1+p

Cycles for a Single SUB Instruction (Using Long Immediate Addressing)

ROM	DARAM	SARAM	External
2	2	2	2+2p

Example 1

```
SUB     DAT80    ;(DP = 8: addresses 0400h–047Fh
```



Example 2

```
SUB     *–,1,AR0  ;(Left shift by 1, SXM = 0)
```

		Before Instruction			After Instruction
ARP		7	ARP		0
AR7		301h	AR7		300h
Data Memory 301h		04h	Data Memory 301h		04h
ACC	X (C)	09h	ACC	1 (C)	01h

Example 3

```
SUB     #8h
```

		Before Instruction			After Instruction
ACC	X (C)	07h	ACC	0 (C)	FFFFFFFFh

Example 4

```
SUB     #0FFFh,4  ;(Left shift by four, SXM = 0)
```

		Before Instruction			After Instruction
ACC	X (C)	0FFFFh	ACC	1 (C)	0Fh

Syntax

SUBB *dma* — Direct addressing
SUBB *ind* [, **AR***n*] — Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SUBB *dma*

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						

SUBB *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) – (data-memory address) – (logical inversion of C) → ACC

Status Bits

Affected by	*Affects*
OVM	OV and C

This instruction is not affected by SXM.

Description

The content of the addressed data-memory location and the logical inversion of the carry bit is subtracted from the accumulator with sign extension suppressed. The carry bit is then affected in the normal manner: the carry bit is cleared (C = 0) if the result of the subtraction generates a borrow; it is set (C = 1) if it does not generate a borrow.

The SUBB instruction can be used in performing multiple-precision arithmetic.

Words

1

Cycles

Cycles for a Single SUBB Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SUBB Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 0400h–047Fh)
```

	Before Instruction		After Instruction
Data Memory 405h	06h	Data Memory 405h	06h
ACC (C = 0)	06h	ACC (C = 0)	0FFFFFFFFh

Example 2

```
SUBB      *
```

	Before Instruction		After Instruction
ARP	6	ARP	6
AR6	301h	AR6	301h
Data Memory 301h	02h	Data Memory 301h	02h
ACC (C = 1)	04h	ACC (C = 1)	02h

In the first example, C is originally zeroed, presumably from the result of a previous subtract instruction that performed a borrow. The effective operation performed was 6 – 6 – (0–) = –1, generating another borrow (resetting carry) in the process. In the second example, no borrow was previously generated (C = 1), and the result from the subtract instruction does not generate a borrow.

Syntax

SUBC *dma*	Direct addressing
SUBC *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SUBC *dma*

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						

SUBC *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

For (ACC) ≥ 0 and (data-memory address) ≥ 0:

Increment PC, then ...
(ACC) – [(data-memory address) × 2^{15}] → ALU output

If ALU output ≥ 0
  Then (ALU output) × 2 + 1 → ACC
  Else (ACC) × 2 → ACC

Status Bits

Affects
OV and C

Description

The SUBC instruction performs conditional subtraction, which can be used for division as follows: Place a positive 16-bit dividend in the low accumulator and clear the high accumulator. Place a 16-bit positive divisor in data memory. Execute SUBC 16 times. After completion of the last SUBC, the quotient of the division is in the lower-order 16 bits of the accumulator, and the remainder is in the higher-order 16 bits of the accumulator. For negative accumulator and/or data-memory values, SUBC cannot be used for division.

If the 16-bit dividend contains fewer than 16 significant bits, the dividend may be placed in the accumulator and left shifted by the number of leading nonsignificant 0s. The number of executions of SUBC is reduced from 16 by that number. One leading 0 is always significant.

SUBC operations performed as stated above are not affected by the sign-extension mode bit (SXM).

SUBC affects OV but is not affected by OVM; therefore, the accumulator does not saturate upon positive or negative overflows when executing this instruction. The carry bit is affected in the normal manner during this instruction: the carry bit is cleared (C = 0) if the result of the subtraction generates a borrow and is set (C = 1) if it does not generate a borrow.

Words 1

Cycles

Cycles for a Single SUBC Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SUBC Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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)
```

	C	Before Instruction		C	After Instruction
Data Memory 302h		01h	Data Memory 302h		01h
ACC	X	04h	ACC	0	08h

Example 2

```
RPT       #15
SUBC      *
```

	C	Before Instruction		C	After Instruction
ARP		3	ARP		3
AR3		1000h	AR3		1000h
Data Memory 1000h		07h	Data Memory 1000h		07h
ACC	X	41h	ACC	1	20009h

Syntax

SUBS *dma*	Direct addressing
SUBS *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SUBS *dma*

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						

SUBS *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) – (data-memory address) → ACC

Status Bits

Affected by	*Affects*
OVM	OV and C

This instruction is not affected by SXM.

Description

The content of the specified data-memory location is subtracted from the accumulator with sign extension suppressed. The data is treated as a 16-bit unsigned number, regardless of SXM. The accumulator behaves as a signed number. SUBS produces the same results as a SUB instruction with SXM = 0 and a shift count of 0.

The carry bit is cleared (C = 0) if the result of the subtraction generates a borrow and is set (C = 1) if it does not generate a borrow.

Words

1

Cycles

Cycles for a Single SUBS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SUBS Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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		0F003h	Data Memory 802h		0F003h
ACC	X (C)	0F105h	ACC	1 (C)	102h

Example 2

```
SUBS      *         ;(SXM = 1)
```

		Before Instruction			After Instruction
ARP		0	ARP		0
AR0		310h	AR0		310h
Data Memory 310h		0F003h	Data Memory 310h		0F003h
ACC	X (C)	0FFFF105h	ACC	1 (C)	0FFF0102h

Syntax

SUBT *dma*	Direct addressing
SUBT *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

SUBT *dma*

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						

SUBT *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(ACC) – [(data-memory address) $\times$ $2^{(TREG(3:0))}$] $\rightarrow$ (ACC)

If SXM = 1
Then (data-memory address) is sign-extended.
If SXM = 0
Then (data-memory address) is not sign-extended.

Status Bits

Affected by	*Affects*
OVM and SXM	OV and C

Description

The data-memory value is left shifted and subtracted from the accumulator. The left shift is defined by the four LSBs of TREG, resulting in shift options from 0 to 15 bits. The result replaces the accumulator contents. Sign extension on the data-memory value is controlled by the SXM status bit.

The carry bit is cleared (C = 0) if the result of the subtraction generates a borrow and is set (C = 1) if it does not generate a borrow.

Words

1

Cycles

Cycles for a Single SUBT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an SUBT Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 = 5: addresses 0280h–02FFh)
```

		Before Instruction			After Instruction
Data Memory 2FFh		06h	Data Memory 2FFh		06h
TREG		08h	TREG		08h
ACC	X	0FDA5h	ACC	1	0F7A5h
	C			C	

Example 2

```
SUBT      *
```

		Before Instruction			After Instruction
ARP		1	ARP		1
AR1		800h	AR1		800h
Data Memory 800h		01h	Data Memory 800h		01h
TREG		08h	TREG		08h
ACC	X	0h	ACC	0	FFFFFF00h
	C			C	

Syntax

TBLR *dma*	Direct addressing
TBLR *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

TBLR *dma*

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						

TBLR *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...

(PC) → MSTACK
(ACC(15:0)) → PC
(pma) → data-memory address
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC

While (repeat counter) ≠ 0
  (pma) → data-memory address
  For indirect, modify (current AR) and (ARP) as specified
  (PC) + 1 → PC
  (repeat counter) –1 → repeat counter.

(MSTACK) → PC

Status Bits

None

Description

The TBLR instruction transfers a word from a location in program memory to a data-memory location specified by the instruction. The program-memory address is defined by the low-order 16 bits of the accumulator. For this operation, a read from program memory is performed, followed by a write to data memory. When repeated with the repeat (RPT) instruction, TBLR effectively becomes a single-cycle instruction, and the program counter that was loaded with (ACC(15:0)) is incremented once each cycle.

Words

1

Cycles

Cycles for a Single TBLR Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of a TBLR Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}+p_{code}$

† 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 of a TBLR Instruction (Continued)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
Source: DARAM/ROM Destination: SARAM	n+2	n+2	n+2 n+4†	n+2+p_{code}
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: addresses 0200h–027Fh)
```

	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

TBLW *dma*	Direct addressing
TBLW *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

TBLW *dma*

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						

TBLW *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(PC+1) → MSTACK
(ACC(15:0)) → PC+1
(data-memory address) → pma,
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC

While (repeat counter) ≠ 0
(data-memory address) → pma,
For indirect, modify (current AR) and (ARP) as specified
(PC) + 1 → PC
(repeat counter) –1 → repeat counter.

(MSTACK) → PC+1

Status Bits

None

Description

The TBLW instruction transfers a word in data memory to program memory. The data-memory address is specified by the instruction, and the program-memory address is specified by the lower 16 bits of the accumulator. A read from data memory is followed by a write to program memory to complete the instruction. When repeated with the repeat (RPT) instruction, TBLW effectively becomes a single-cycle instruction, and the program counter that was loaded with (ACC(15:0)) is incremented once each cycle.

Words

1

Cycles

Cycles for a Single TBLW Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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	$3+d_{src}$	$3+d_{src}$	$3+d_{src}$	$3+d_{src}+p_{code}$
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	$3+d_{src}$	$3+d_{src}$	$3+d_{src}$ $4+d_{src}$†	$3+d_{src}+p_{code}$
Source: DARAM/ROM 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 of a TBLW Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}+p_{code}$

† 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

Cycles for a Repeat (RPT) Execution of a TBLW Instruction (Continued)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
Source: DARAM/ROM Destination: SARAM	n+2	n+2	n+2 n+3†	$n+2+p_{code}$
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/ROM 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: addresses 1000h–107Fh)
```

	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 $\rightarrow$ stack
22h $\rightarrow$ PC

Status Bits Not affected by INTM; does not affect INTM.

Description The TRAP instruction is a software interrupt that transfers program control to program-memory location 22h and pushes the program counter (PC) plus 1 onto the hardware stack. The instruction at location 22h may contain a branch instruction to transfer control to the TRAP routine. Putting (PC + 1) onto the stack enables a return instruction to pop the return address (which points to the instruction after TRAP) from the stack. The TRAP instruction is not maskable.

Words 1

Cycles

Cycles for a Single TRAP Instruction

ROM	DARAM	SARAM	External
4	4	4	4+3p†

† The processor performs speculative fetching by reading two additional instruction words. If the PC discontinuity is taken, these two instruction words are discarded.

Example

```
TRAP        ;PC + 1 is pushed onto the stack, and then
            ;control is passed to program memory location
            ;22h.
```

Syntax

XOR *dma*	Direct addressing
XOR *ind* [, **AR***n*]	Indirect addressing
XOR #*lk* [, *shift*]	Long immediate addressing
XOR #*lk*,**16**	Long immediate with left shift of 16

Operands

dma:	7 LSBs of the data-memory address
shift:	Left shift value from 0 to 15 (defaults to 0)
n:	Value from 0 to 7 designating the next auxiliary register
lk:	16-bit long immediate value
ind:	Select one of the following seven options:
	* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

XOR *dma*

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						

XOR *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

XOR #*lk* [, *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	shift			
lk															

XOR #*lk*, **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
lk															

Execution

Increment PC, then ...

Event(s)	*Addressing mode*
(ACC(15:0)) XOR (data-memory address) $\rightarrow$ ACC(15:0) (ACC(31:16)) $\rightarrow$ ACC(31:16)	Direct or indirect
(ACC(31:0)) XOR lk $\times$ 2^{shift} $\rightarrow$ ACC(31:0)	Long immediate
(ACC(31:0)) XOR lk $\times$ 2^{16} $\rightarrow$ ACC(31:0)	Long immediate with left shift of 16

Status Bits None

Description With direct or indirect addressing, the low half of the accumulator value is exclusive ORed with the content of the addressed data memory location, and the result replaces the low half of the accumulator value; the upper half of the accumulator value is unaffected. With immediate addressing, the long immediate constant is shifted and zero filled on both ends and exclusive ORed with the entire content of the accumulator. The carry bit (C) is unaffected by XOR.

Words

Words	*Addressing mode*
1	Direct or indirect
2	Long immediate

Cycles

Cycles for a Single XOR Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of an XOR Instruction (Using Direct and Indirect Addressing)

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 XOR Instruction (Using Long Immediate Addressing)

ROM	**DARAM**	**SARAM**	**External**
2	2	2	2+2p

Example 1

```
XOR   DAT127        ;(DP = 511: addresses FF80h-FFFFh)
```

		Before Instruction			After Instruction
Data Memory 0FFFFh		0F0F0h	Data Memory 0FFFFh		0F0F0h
ACC	X (C)	12345678h	ACC	X (C)	1234A688h

Example 2

```
XOR   *+,AR0
```

		Before Instruction			After Instruction
ARP		7	ARP		0
AR7		300h	AR7		301h
Data Memory 300h		0FFFFh	Data Memory 300h		0FFFFh
ACC	X (C)	1234F0F0h	ACC	X (C)	12340F0Fh

Example 3

```
XOR   #0F0F0h,4     ;(First shift data value left by
                    ;four)
```

		Before Instruction			After Instruction
ACC	X (C)	11111010h	ACC	X (C)	111E1F10h

Syntax

ZALR *dma*	Direct addressing
ZALR *ind* [, **AR***n*]	Indirect addressing

Operands

dma: 7 LSBs of the data-memory address
n: Value from 0 to 7 designating the next auxiliary register
ind: Select one of the following seven options:
* *+ *– *0+ *0– *BR0+ *BR0–

Opcode

ZALR *dma*

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						

ZALR *ind* [, **AR***n*]

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	ARU			N	NAR		

Note: ARU, N, and NAR are defined in section 6.3, *Indirect Addressing Mode* (page 6-9).

Execution

Increment PC, then ...
(data-memory address) → ACC(31:16)
8000h → ACC(15:0)

Status Bits

None

Description

The ZALR instruction loads a 16-bit data-memory value into the high word of the accumulator. The instruction rounds the value by adding half of the value of the LSB: bit 15 of the accumulator is set, and bits 14 are cleared.

Words

1

Cycles

Cycles for a Single ZALR Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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 of a ZALR Instruction

	Program			
Operand	**ROM**	**DARAM**	**SARAM**	**External**
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: addresses 1000h–107Fh)
```

	Before Instruction		After Instruction
Data Memory 1003h	3F01h	Data Memory 1003h	3F01h
ACC (C: X)	77FFFFh	ACC (C: X)	3F018000h

Example 2

```
ZALR      *–,AR4
```

	Before Instruction		After Instruction
ARP	7	ARP	4
AR7	0FF00h	AR7	0FEFFh
Data Memory 0FF00h	0E0E0h	Data Memory 0FF00h	0E0E0h
ACC (C: X)	107777h	ACC (C: X)	0E0E08000h

Chapter 8

On-Chip Peripherals

This chapter discusses on-chip peripherals connected to the ’C20x CPU and their control registers. The on-chip peripherals are controlled through I/O mapped registers. The operations of the timer and the serial ports are synchronized to the processor through interrupts and interrupt polling. The ’C20x on-chip peripherals are:

- Clock generator
- Timer
- Software-programmable wait-state generator
- General-purpose I/O pins
- Synchronous serial port (SSP)
- Asynchronous serial port (ASP), or UART

The serial ports are discussed in Chapter 9 and Chapter 10.

For examples of program code for the on-chip peripherals, see Appendix D, *Program Examples*.

Topic **Page**

8.1 Control of On-Chip Peripherals 8-2

8.2 Clock Generator 8-4

8.3 CLKOUT1-Pin Control (CLK) Register 8-7

8.4 Timer 8-8

8.5 Wait-State Generator 8-15

8.6 General-Purpose I/O Pins 8-18

8.1 Control of On-Chip Peripherals

The on-chip peripherals are controlled by accessing control registers that are mapped to on-chip I/O space. Data is also transferred to and from the peripherals through these registers. Setting and clearing bits in these registers can enable, disable, initialize, and dynamically reconfigure the on-chip peripherals.

On a device reset, the CPU sends an internal $\overline{\text{SRESET}}$ signal to the peripheral circuits. Table 8–1 lists the peripheral registers and summarizes what happens when the values in these registers are reset. For a description of all the effects of a device reset, see section 5.7, *Reset Operation*, on page 5-35.

Table 8–1. Peripheral Register Locations and Reset Conditions

Register Name	I/O Address		Reset Value	Effects at Reset
	’C209	Other ’C20x		
PMST	–	FFE4h	000xh	*Program memory status register*. SARAM mapped into program and data memory. MP/$\overline{\text{MC}}$ and LEVEXT8 bits depend on external pin state.
CLK	–	FFE8h	0000h	*CLKOUT1-pin control (CLK) register.* The CLKOUT1 signal is available at the CLKOUT1 pin.
SDTR	–	FFF0h	xxxxh	*Synchronous data transmit and receive register.* The value in this register is undefined after reset.
SSPCR	–	FFF1h	0030h	*Synchronous serial port control register.* The port emulation mode is set to immediate stop. Error and status flags are reset. Receive interrupts are set to occur when the receive buffer is not empty. Transmit interrupts are set to occur when the transmit buffer can accept one or more words. External clock and frame synchronization sources are selected. Continuous mode is selected. Digital loopback mode is disabled. The receiver and transmitter are enabled.
SSPST	–	FFF2h	0000h	*Synchronous serial port status register*. Data word size is 16 bits. Sign extension is off. FIFO registers are empty. Clock prescaler is disabled. Input clock is CLKOUT1. CLKX polarity is normal. FSX rate is rate at which data is written to transmit FIFO.
SSPMC	–	FFF3h	0000h	*Synchronous serial port multichannel register.* GPC is disabled. Multichannel mode is disabled. SPI mode is disabled.
ADTR	–	FFF4h	xxxxh	*Asynchronous data transmit and receive register.* The value in this register is undefined after reset.

Table 8–1. Peripheral Register Locations and Reset Conditions (Continued)

Register Name	I/O Address		Reset Value	Effects at Reset
	’C209	Other ’C20x		
ASPCR	–	FFF5h	0000h	*Asynchronous serial port control register.* The port emulation mode is set to immediate stop. Receive, transmit, and delta interrupts are disabled. One stop bit is selected. Auto-baud alignment is disabled. The TX pin is forced high between transmissions. I/O pins IO0, IO1, IO2, and IO3 are configured as inputs. The port is disabled.
IOSR	–	FFF6h	18xxh	*I/O status register.* Auto-baud alignment is disabled. Error and status flags are reset. The lower eight bits are dependent on the values on pins IO0, IO1, IO2, and IO3 at reset.
BRD	–	FFF7h	0001h	*Baud rate divisor register.* A baud rate of (CLKOUT1 rate)/16 is selected.
TCR	FFFCh	FFF8h	0000h	*Timer control register.* The divide-down value is 0, and the timer is started.
PRD	FFFDh	FFF9h	FFFFh	*Timer period register.* The next value to be loaded into the timer counter register (TIM) is at its highest value.
TIM	FFFEh	FFFAh	FFFFh	*Timer counter register.* The timer count is at its highest value.
SSPCT	–	FFFBh	0000h	*Synchronous serial port counter register*. SSP counter bits are 0.
WSGR	FFFFh	FFFCh	0FFFh	*Wait-state generator control register.* The maximum number of wait states are selected for off-chip program, data, and I/O spaces.

8.2 Clock Generator

The high pulse of the master clock output signal (CLKOUT1) signifies the logic phase of the device (the phase when values are changed), while the low pulse signifies the latch phase (the phase when values are latched). CLKOUT1 determines much of the device's operational speed. For example:

- ❑ The timer clock rate is a fraction of the rate of CLKOUT1.
- ❑ Each instruction cycle is equal to one CLKOUT1 period.
- ❑ Each wait state generated by the READY signal or by the on-chip wait-state generator is equal to one CLKOUT1 period.

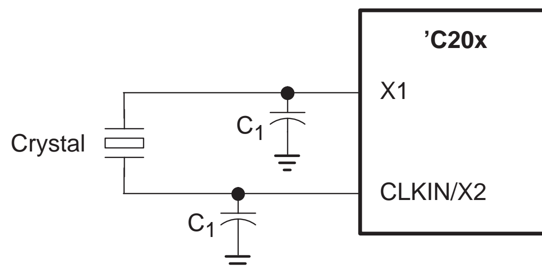
You control the rate of CLKOUT1 with the on-chip clock generator. The clock generator creates an internal CPU clock signal CLKOUT1 whose rate is a fraction or multiple of a source clock signal CLKIN. This generator consists of two independent components, an oscillator and a phase lock loop (PLL) circuit. The internal oscillator, in conjunction with an external resonator circuit, allows you to generate CLKIN internally and create a CLKOUT1 signal that oscillates at a multiple (0.5, 1, 2, or 4 times) of the frequency of CLKIN. The PLL makes the rate of CLKOUT1 a multiple of the rate of CLKIN and locks the phase of CLKOUT1 to that of CLKIN.

CLKIN can be generated by the internal oscillator or by an external oscillator:

- ❑ **Internal oscillator**. The clock source is generated internally by connecting a crystal resonator circuit across the CLKIN/X2 and X1 pins. The crystal should be in either fundamental or overtone operation and parallel resonant, with an effective series resistance of 30 ohms and a power dissipation of 1 mW. It should also be specified at a load capacitance of 20 pF. Figure 8–1 shows the setup for a fundamental frequency crystal. Overtone crystals require an additional tuned-LC circuit.

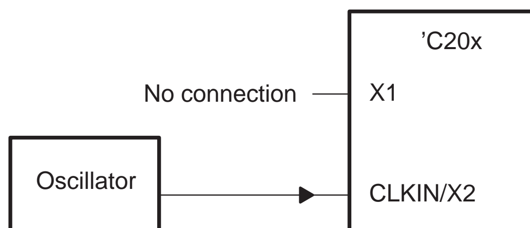
  If the internal oscillator is used, the frequency of CLKOUT1 is half the oscillating frequency of the crystal in ÷2 mode. For example, a 40-MHz crystal will provide a CLKOUT1 rate of 20 MHz, providing 20 MIPS of processing power.

Figure 8–1. Using the Internal Oscillator



- **External Oscillator**. If an external oscillator is used, its output must be connected to the CLKIN/X2 pin. The X1 pin must be left unconnected. See Figure 8–2.

Figure 8–2. Using an External Oscillator



Regardless of the method used to generate CLKOUT1, CLKOUT1 is also available at the CLKOUT1 pin, unless the pin is turned off by the CLK register (see section 8.3).

You can lower the power requirements for the ’C20x by slowing down or stopping the input clock.

Note:

When restarting the system, activate $\overline{\text{RS}}$ before starting or stopping the clock, and hold it active until the clock stabilizes. This brings the device back to a known state.

8.2.1 Clock Generator Options

The ’C20x provides four clock modes: divide-by-2 (÷2), multiply-by-1 (×1), multiply-by-2 (×2), and multiply-by-4 (×4). The ÷2 mode operates the CPU at half the input clock rate. Each of the other modes operates the CPU at a multiple of the input clock rate and phase locks the output clock with the the input clock. You set the mode by changing the levels on the DIV1 and DIV2 pins. For each mode, Table 8–2 shows the generated CPU clock rate and the state of DIV2, DIV1, the internal oscillator, and the internal phase lock loop (PLL).

Notes:

1) Change DIV1 and DIV2 only while the reset signal ($\overline{\text{RS}}$) is active.

2) The PLL requires approximately 2500 cycles to lock the output clock signal to the input clock signal. When setting the ×1, ×2, or ×4 mode, keep the reset ($\overline{\text{RS}}$) signal active until at least three cycles after the PLL has stabilized.

Table 8–2. ’C20x Input Clock Modes

Clock Mode	CLKOUT1 Rate	DIV2	DIV1	External CLKIN Source?	Internal Oscillator	Internal PLL
÷ 2	CLKOUT1 = CLKIN ÷ 2	0	0	No	Enabled	Disabled
				Yes	Disabled	Disabled
× 1	CLKOUT1 = CLKIN × 1	0	1	Required	Disabled	Enabled
× 2	CLKOUT1 = CLKIN × 2	1	0	Required	Disabled	Enabled
× 4	CLKOUT1 = CLKIN × 4	1	1	Required	Disabled	Enabled

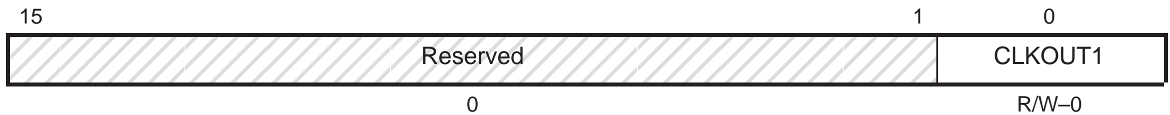
Remember the following when configuring the clock mode:

- The clock mode configuration cannot be dynamically changed. After you change the levels on DIV1 and DIV2, the mode is not changed until a hardware reset is executed ($\overline{RS}$ low).
- The operation of the PLL circuit is affected by the operating voltage of the device. If your device operates at 5V, the PLL5V signal should be tied high at the PLL5V pin. If you have a 3-V device, tie PLL5V low.
- The ×1, ×2, and ×4 modes use an internal phase lock loop (PLL) that requires approximately 2500 cycles to lock. Delay the rising edge of $\overline{RS}$ until at least three cycles after the PLL has stabilized. When the PLL is used, the duty cycle of the CLKIN signal is more flexible, but the minimum duty cycle should not be less than 10 nanoseconds. When the PLL is not used, no phase-locking time is necessary, but the minimum pulse width must be 45% of the minimum clock cycle.

8.3 CLKOUT1-Pin Control (CLK) Register

You can use bit 0 of the CLK register to turn off the pin for the master clock output signal (CLKOUT1). The CLK register is located at address FFE8h in I/O space and has the organization shown in Figure 8–3.

Figure 8–3. ’C20x CLK Register — I/O-Space Address FFE8h



Note: 0 = Always read as zeros; R = Read access; W = Write access; value following dash (–) is value after reset.

If the CLKOUT1 bit is 1, the CLKOUT1 signal is not available at the CLKOUT1 pin; if the bit is 0, CLKOUT1 is available at the pin. At reset, this bit is cleared to 0. When the IDLE instruction puts the CPU into a power-down mode, CLKOUT1 remains active at the pin if the CLKOUT1 bit is 0. (For more information on the ’C20x power-down mode, see section 5.8, *Power-Down Mode*, on page 5-40).

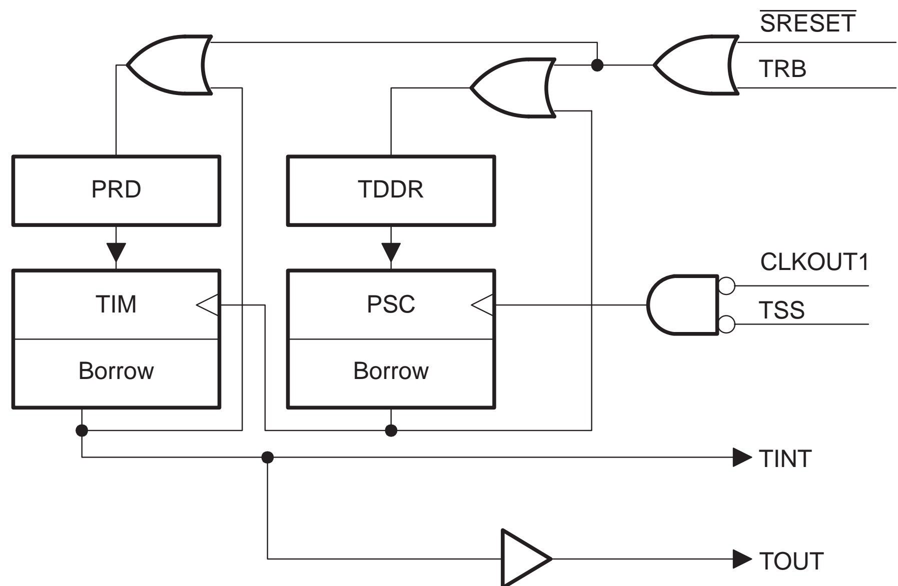
For the current status of CLKOUT1, read bit 0. To change the status, write to bit 0. When programming, allow the CLKOUT1 pin two cycles to change its state from on to off or from off to on. Bits 15–1 are reserved and are always read as 0s.

8.4 Timer

The ’C20x features an on-chip timer with a 4-bit prescaler. This timer is a down counter that can be stopped, restarted, reset, or disabled by specific status bits. You can use the timer to generate periodic CPU interrupts.

Figure 8–4 shows a functional block diagram of the timer. There is a 16-bit main counter (TIM) and a 4-bit prescaler counter (PSC). The TIM is reloaded from the period register PRD. The PSC is reloaded from the period register TDDR.

Figure 8–4. Timer Functional Block Diagram



Each time a counter decrements to zero, a borrow is generated on the next CLKOUT1 cycle, and the counter is reloaded with the contents of its corresponding period register. The contents of the PRD are loaded into the TIM when the TIM decrements to 0 or when a 1 is written to the timer reload bit (TRB) in the timer control register (TCR). Similarly, the PSC is loaded with the value in the TDDR when the PSC decrements to 0 or when a 1 is written to TRB.

When the TIM decrements to 0, it generates a borrow pulse that has a duration equal to that of a CLKOUT1 cycle ($t_{c(C)}$). This pulse is sent to:

- The external timer output (TOUT) pin
- The CPU, as a timer interrupt (TINT) signal

The TINT request automatically sets the TINT flag bit in the interrupt flag register (IFR). You can mask or unmask the request with the interrupt mask register (IMR). If you are not using the timer, mask TINT so that it does not cause an unexpected interrupt.

8.4.1 Timer Operation

Here is a typical sequence of events for the timer:

1) The PSC decrements on each succeeding CLKOUT1 pulse until it reaches 0.

2) On the next CLKOUT1 cycle, the TDDR loads the new divide-down count into the PSC, and the TIM decrements by 1.

3) The PSC and the TIM continue to decrement in the same way until the TIM decrements to 0.

4) On the next CLKOUT1 cycle, a timer interrupt (TINT) is sent to the CPU, a pulse is sent to the TOUT pin, the new timer count is loaded from the PRD into the TIM, and the PSC is decremented once.

The TIM decrements by one every (TDDR+1) CLKOUT1 cycles. When PRD, TDDR, or both are nonzero, the timer interrupt rate is defined by Equation 8–1, where $t_{c(CO)}$ is the period of CLKOUT1, u is the TDDR value plus 1, and v is the PRD value plus 1. When PRD = TDDR = 0, the timer interrupt rate is (CLKOUT1 rate)/2.

Equation 8–1. Timer Interrupt Rate for Nonzero TDDR and/or PRD

$$\text{TINT rate} = \frac{1}{t_{c(CO)}} \times \frac{1}{u \times v} = \frac{1}{t_{c(CO)}} \times \frac{1}{(\text{TDDR} + 1) \times (\text{PRD} + 1)} = \frac{\text{CLKOUT1 rate}}{(\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

Note:

Equation 8–1 is not valid for TDDR = PRD = 0; in this case, the timer interrupt rate defaults to (CLKOUT1 rate)/2.

In Equation 8–1 the timer interrupt rate equals the CLKOUT1 frequency ($1/t_{c(CO)}$) divided by two independent factors (u and v). Each of the two divisors is implemented with a down counter and a period register. See the timer functional block diagram, Figure 8–4, on page 8-8. The counter and period registers for the divisor u are the PSC and TDDR, respectively, both 4-bit fields of the timer control register (TCR). The counter and period registers for the divisor v are the TIM and PRD, respectively. Both are16-bit registers mapped to I/O space.

The 4-bit TDDR (timer divide-down register) and the 4-bit PSC (prescaler counter) are contained in the timer control register (TCR) described in section 8.4.2. The TIM (timer counter register) and the PRD (timer period register) are 16-bit registers described in section 8.4.3. You can read the TCR, TIM, and PRD to obtain the current status of the timer and its counters.

Note:

Read the TIM for the current value in the timer. Read the TCR for the PSC value. Because it takes two instructions to read both the TIM and the TCR, the PSC may decrement between the two reads, making comparison of the reads inaccurate. Therefore, where precise timing measurements are necessary, you may want to stop the timer before reading the two values. (Set the TSS bit of the TCR to 1 to stop the timer; clear TSS to 0 to restart the timer.)

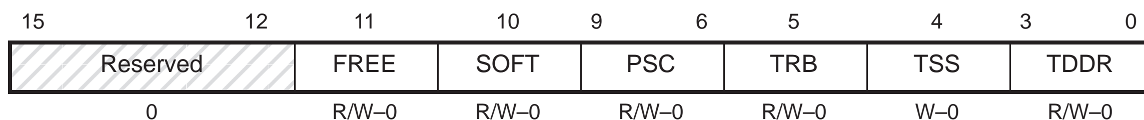
8.4.2 Timer Control Register (TCR)

The TCR, a 16-bit register mapped to on-chip I/O space, contains the control bits that:

- Control the mode of the timer
- Specify the current count in the prescaler counter
- Reload the timer
- Start and stop the timer
- Define the divide-down value of the timer

For ’C20x devices other than the ’C209, Figure 8–5 shows the bit layout of the TCR. Descriptions of the bits follow the figure. For a description of the ’C209 TCR, see section 11.4.2 on page 11-16.

Figure 8–5. ’C20x Timer Control Register (TCR) — I/O-Space Address FFF8h



Note: 0 = Always read as zeros; R = Read access; W = Write access; value following dash (–) is value after reset.

Table 8–3. ’C20x TCR — I/O Space Address FFF8h Bit Descriptions

Bit No.	Name	Function
15–12	Reserved	Bits 15–12 are reserved and are always read as 0s.
11–10	FREE, SOFT	These bits are special emulation bits that determine the state of the timer 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 timer continues to run (that is, runs free). In this case, SOFT is a *don’t care*. But if FREE is 0, then SOFT takes effect. In this case, if SOFT = 0, the timer halts the next time the TIM decrements. If the SOFT bit is 1, then the timer halts when the TIM has decremented to zero. The default (reset) setting is FREE = 0 and SOFT = 0. The available run and emulation modes are: 0 0 Stop after the next decrement of the TIM (hard stop) 0 1 Stop after the TIM decrements to 0 (soft stop) 1 0 Free run 1 1 Free run
9–6	PSC	Timer prescaler counter. These four bits hold the current prescale count for the timer. For every CLKOUT1 cycle that the PSC value is greater than 0, the PSC decrements by one. One CLKOUT1 cycle after the PSC reaches 0, the PSC is loaded with the contents of the TDDR, and the timer counter register (TIM) decrements by one. The PSC is also reloaded whenever the timer reload bit (TRB) is set by software. The PSC can be checked by reading the TCR, but it cannot be set directly. It must get its value from the timer divide-down register (TDDR). At reset, the PSC is set to 0.
5	TRB	Timer reload bit. When you write a 1 to TRB, the TIM is loaded with the value in the PRD, and the PSC is loaded with the value in the timer divide-down register (TDDR). The TRB bit is always read as zero.

Table 8–3. ’C20x TCR — I/O Space Address FFF8h Bit Descriptions (Continued)

Bit No.	Name	Function
4	TSS	Timer stop status bit. TSS stops or starts the timer. At reset, TSS is cleared to 0 and the timer immediately starts. 0 Starts or restarts the timer. 1 Stops the timer.
3–0	TDDR	Timer divide-down register. Every (TDDR + 1) CLKOUT1 cycles, the timer counter register (TIM) decrements by one. At reset, the TDDR bits are cleared to 0. If you want to increase the overall timer count by an integer factor, write this factor minus one to the four TDDR bits. When the prescaler counter (PSC) value is 0, one CLKOUT1 cycle later, the contents of the TDDR reload the PSC, and the TIM decrements by one. TDDR also reloads the PSC whenever the timer reload bit (TRB) is set by software.

8.4.3 Timer Counter Register (TIM) and Timer Period Register (PRD)

These two registers work together to provide the current count of the timer:

- The 16-bit timer counter register (TIM) holds the current count of the timer. The TIM decrements by one every (TDDR+1) CLKOUT1 cycles. When the TIM decrements to zero, the TINT bit of the interrupt flag register (IFR) is set (causing a pending timer interrupt), and a pulse is sent to the TOUT pin.

  You cannot directly write to the TIM register. At reset, this register is set to hold its maximum value of FFFFh. See Table 8–1 (page 8-2) for the address of this register.

- The 16-bit timer period register (PRD) holds the next starting count for the timer. When the TIM decrements to zero, in the following cycle, the contents of the PRD are loaded into the TIM. The PRD contents are also loaded into the TIM when you set the timer reload bit (TRB).

  You can program the PRD to contain a value from 0 to 65 535 (FFFFh). After reset, the PRD holds its maximum value of FFFFh. See Table 8–1 (page 8-2) for the address of this register. If you are not using the timer, you can mask TINT and then use the PRD as a general-purpose data-memory location.

You control the timer’s current and next periods. You can write to or read from the TIM and PRD on any cycle. You can monitor and control the count by reading from the TIM and writing the next counter period to the PRD without disturbing the current timer count. The timer will start the next period after the current

count is complete. If you use TINT, you should program the PRD and TIM before unmasking TINT, to avoid unwanted interrupts.

Once a reset is initiated, the TIM begins to decrement only after reset is deasserted.

8.4.4 Setting the Timer Interrupt Rate

When the divide-down value (TDDR) is 0, you can program the timer to generate an interrupt (TINT) every 2 to 65 536 cycles by programming the period register (PRD) from 0 to 65 535 (FFFFh). When TDDR is nonzero (1 to 15), the timer interrupt rate decreases.

If TDDR, PRD, or both are nonzero, the timer interrupt rate is given by:

$$\text{TINT rate} = \frac{\text{CLKOUT1 rate}}{(\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

Note:

When TDDR = PRD = 0, the timer interrupt rate defaults to (CLKOUT1 rate)/2.

As an example of setting the timer interrupt rate, suppose the CLKOUT1 rate is 10 MHz and you want to use the timer to generate a clock signal with a rate of 10 kHz. You need to divide the CLKOUT1 rate by 1000. The TDDR is loaded with 4, so that every 5 CLKOUT1 cycles, the TIM decrements by one. The PRD is loaded with the starting count (199) for the TIM. These values are verified with the TINT rate equation:

$$\text{TINT rate} = \text{CLKOUT1 rate} \times \frac{1}{(\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

$$\text{TINT rate} = \frac{1 \text{ CLKOUT1 cycle}}{0.10 \times 10^{-6} \text{ s}} \times \frac{1 \text{ TINT cycle}}{(4 + 1) \times (199 + 1) \text{ CLKOUT1 cycles}}$$

$$\text{TINT rate} = \frac{10 \times 10^{3} \text{ TINT cycles}}{\text{s}} = 10 \text{ kHz}$$

The PSC and the TIM would be loaded with the values from the TDDR and the PRD, respectively. Then, one CLKOUT1 cycle after the TIM decrements to 0, the timer would send an interrupt to the CPU.

8.4.5 The Timer at Hardware Reset

On a device reset, the CPU sends an $\overline{\text{SRESET}}$ signal to the peripheral circuits, including the timer. The $\overline{\text{SRESET}}$ signal has the following consequences on the timer:

- The registers TIM and PRD are loaded with their maximum values (FFFFh).
- All the bits of the TCR are cleared to zero with the following results:
  - The divide-down value is 0 (TDDR = 0 and PSC = 0).
  - The timer is started (TSS = 0).
  - The FREE and SOFT bits are both 0.

8.5 Wait-State Generator

Wait states are necessary when you want to interface the ’C20x with slower external logic and memory. By adding wait states, you lengthen the time the CPU waits for external memory or an external I/O port to respond when the CPU reads from or writes to that memory or port. Specifically, the CPU waits one extra cycle (one CLKOUT1 cycle) for every wait state. The wait states operate on CLKOUT1 cycle boundaries.

To avoid bus conflicts, writes from the ’C20x always take at least two CLKOUT1 cycles.

The ’C20x offers two options for generating wait states:

- The READY signal. With the READY signal, you can externally generate any number of wait states.
- The on-chip wait-state generator. With this generator, you can generate zero to seven wait states.

8.5.1 Generating Wait States With the READY Signal

When READY is low, the ’C20x waits one CLKOUT1 cycle and checks READY again. The ’C20x will not continue executing until READY is driven high; therefore, if the READY signal is not used, it should be pulled high during external accesses.

Again, the READY pin can be used to generate any number of wait states. However, even when the ’C20x operates at full speed, it may not respond fast enough to provide a READY-based wait state for the first cycle. For extended wait states using external READY logic, the on-chip wait-state generator should be programmed to generate at least one wait state.

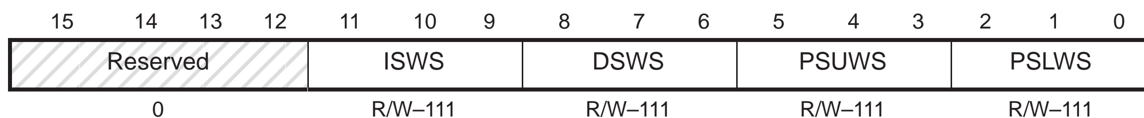
The READY pin has no effect on accesses to *internal* memory or I/O registers, except in the case of the ’C209 (refer to section 11.2, *’C209 Memory and I/O Spaces)*. For a ’C20x device with a bootloader, READY must be high at boot time.

8.5.2 Generating Wait States With the ’C20x Wait-State Generator

For devices other than the ’C209, the software wait-state generator can be programmed to generate zero to seven wait states for a given off-chip memory space (lower program, upper program, data, or I/O), regardless of the state of the READY signal. This wait-state generator has the bit fields shown in Figure 8–6 and described after the figure. For a description of the ’C209 wait-

state generator, see section 11.4.3 on page 11-17. To avoid bus conflicts, all writes to external addresses take at least two cycles. Once the wait-state generator has no zero value, the wait states are extended for both read and write cycles.

Figure 8–6. 'C20x Wait-State Generator Control Register (WSGR) — I/O-Space Address FFFCh



Note: 0 = Always read as zeros; R = Read access; W = Write access; value following dash (–) is value after reset.

Table 8–4. 'C20x WSGR — I/O Space Address FFFCh Bit Descriptions

Bit No.	Name	Function
15–12	Reserved	Bits 15–12 are reserved and are always read as 0s.
11–9	ISWS	I/O-space wait-state bits. Bits 9–11 determine the number of wait states (0, 1, 2, 3, 4, 5, 6, or 7) that are applied to reads from and writes to off-chip I/O space. At reset, the three ISWS bits become 111, setting seven wait states for reads from and writes to off-chip I/O space.
8–6	DSWS	Data-space wait-state bits. Bits 6–8 determine the number of wait states (0, 1, 2, 3, 4, 5, 6, or 7) that are applied to reads from and writes to off-chip data space. At reset, the three DSWS bits become 111, setting seven wait states for reads from and writes to off-chip data space.
5–3	PSUWS	Upper program-space wait-state bits. Bits 3–5 determine the number of wait states (0, 1, 2, 3, 4, 5, 6, or 7) that are applied to reads from and writes to off-chip *upper* program addresses 8000h–FFFFh. At reset, the three PSUWS bits become 111, setting seven wait states for reads from and writes to off-chip upper program space.
2–0	PSLWS	Lower program-space wait-state bits. Bits 0–2 determine the number of wait states (0, 1, 2, 3, 4, 5, 6, or 7) that are applied to reads from and writes to off-chip *lower* program addresses 0h–7FFFh. At reset, the three PSLWS bits become 111, setting seven wait states for reads from and writes to off-chip lower program space.

Table 8–5 shows how to set the number of wait states you want for each type of off-chip memory. For example, if you write 1s to bits 0 through 5, the device will generate seven wait states for off-chip lower program memory and seven wait states for off-chip upper program memory.

Table 8–5. Setting the Number of Wait States With the ’C20x WSGR Bits

ISWS Bits			I/O Wait States	DSWS Bits			Data Wait States	PSUWS Bits			Upper Program Wait States	PSLWS Bits			Lower Program Wait States
11	**10**	**9**		**8**	**7**	**6**		**5**	**4**	**3**		**2**	**1**	**0**	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	2	0	1	0	2	0	1	0	2	0	1	0	2
0	1	1	3	0	1	1	3	0	1	1	3	0	1	1	3
1	0	0	4	1	0	0	4	1	0	0	4	1	0	0	4
1	0	1	5	1	0	1	5	1	0	1	5	1	0	1	5
1	1	0	6	1	1	0	6	1	1	0	6	1	1	0	6
1	1	1	7	1	1	1	7	1	1	1	7	1	1	1	7

In summary, the wait-state generator inserts zero to seven wait states to a given memory space, depending on the values of PSLWS, PSUWS, DSWS, and ISWS, while the READY signal remains high. The READY signal may then be driven low to generate additional wait states. At reset, all WSGR bits are set to 1, making seven wait states the default for every memory space.

8.6 General-Purpose I/O Pins

The ’C20x provides pins that can be used to supply input signals from an external device or output signals to an external device. These pins are not bound to specific uses; rather, they can provide input or output signals for a great variety purposes. You have access to the general-purpose input pin $\overline{BIO}$ and the general-purpose output pin XF. On ’C20x devices other than the ’C209, you also have the pins IO0, IO1, IO2, and IO3, which can each be configured as an input pin or an output pin.

8.6.1 Input Pin $\overline{BIO}$

The general-purpose input pin $\overline{BIO}$ pin provides input from an external device and is particularly helpful as an alternative to an interrupt when time-critical loops must not be disturbed. The $\overline{BIO}$ signal gives you control through three instructions, a conditional branch (BCND), a conditional call (CC), and a conditional return (RETC). Here is an example of each:

- BCND *pma*, BIO

  pma is a program memory address that you specify. The CPU branches to the program memory address if $\overline{BIO}$ is low.

- CC *pma*, BIO

  pma is a program memory address that you specify. If $\overline{BIO}$ is low, the CPU stores the return address to the top of the hardware stack and then branches to the program memory address.

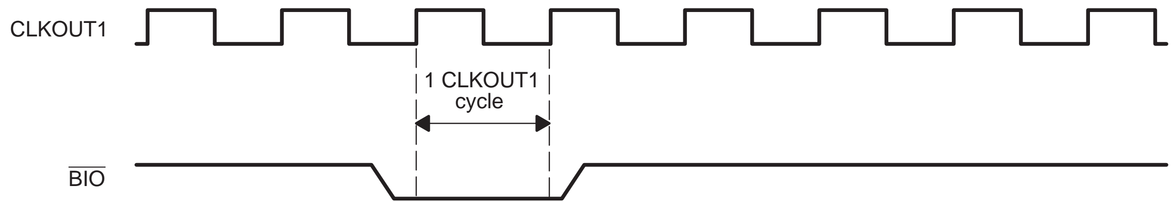
- RETC BIO

  If $\overline{BIO}$ is low, the CPU transfers the return address from the stack to the program counter (PC) to return from a subroutine or interrupt service routine.

If $\overline{BIO}$ is not used, it should be pulled high so that a conditional branch, call, or return will not be executed accidentally.

An example of $\overline{BIO}$ timing is shown in Figure 8–7. This timing diagram is for a sequence of single-cycle, single-word instructions located in external memory. $\overline{BIO}$ must be asserted low for at least one CLKOUT1 cycle. The BCND, CC, and RETC instructions sample the $\overline{BIO}$ pin during their execute phase in the pipeline. Actual timing may vary with different instruction sequences.

Figure 8–7. $\overline{BIO}$ Timing Diagram Example



8.6.2 Output Pin XF

The XF pin is the external flag output pin. If you connect XF to an input pin of another processor, you can use XF as a signal to other processor. The most recent XF value is latched in the ’C20x, and that value is indicated by the XF status bit of status register ST1. You can set XF (XF = 1) with the SETC XF (set external flag) instruction and clear it (XF = 0) with the CLRC XF (clear external flag) instruction. In addition, you can write to ST1 with the LST (load status register) instruction. During a hardware reset, XF is set to 1.

8.6.3 Input/Output Pins IO0, IO1, IO2, and IO3

For additional input/output control, ’C20x devices other than the ’C209 have pins IO0, IO1, IO2, and IO3, which can be individually configured as inputs or outputs. These pins are software-controllable with the asynchronous serial port control register (ASPCR) and the I/O status register (IOSR). For the details of configuring and using these I/O pins, see section 10.3.5, *Using I/O Pins IO3, IO2, IO1, and IO0*, on page 10-15.

Chapter 9

Synchronous Serial Port

The ’C20x devices have a synchronous serial port that provides direct communication with serial devices such as codecs (coder/decoders) and serial A/D converters. The serial port may also be used for intercommunication between processors in multiprocessing applications.

The synchronous serial port offers these features:

- Two four-word-deep FIFO buffers
- Interrupts generated by the FIFO buffers
- A wide range of speeds of operation
- Burst and continuous modes of operation

For examples of program code for the synchronous serial port, see Appendix D, *Program Examples*.

Topic **Page**

9.1 Overview of the Synchronous Serial Port 9-2
9.2 Components and Basic Operation 9-3
9.3 Controlling and Resetting the Port 9-8
9.4 Managing the Contents of the FIFO Buffers 9-15
9.5 Transmitter Operation 9-16
9.6 Receiver Operation 9-22
9.7 Troubleshooting 9-25
9.8 Enhanced Synchronous Serial Port (ESSP) 9-29
9.9 ESSP Pins 9-30
9.10 ESSP Registers 9-32
9.11 ESSP Register Programming Considerations 9-40

9.1 Overview of the Synchronous Serial Port

Both receive and transmit sections of the synchronous serial port have a four-word-deep first-in, first-out (FIFO) buffer. The FIFO buffers reduce the amount of CPU overhead inherent in servicing transmit or receive data by reducing the number of transmit or receive interrupts that occur during a transfer. The synchronous serial port is reset 16 CLKOUT1 cycles after the rising edge of the pin, during device reset.

In the internal clock mode, the maximum transmission rate for both transmit and receive operations is the CPU clock rate divided by two, or (CLKOUT1 rate)/2. Therefore, the maximum rate is 10 megabits/s for a 20-MHz (50-ns) device, 14.28 megabits/s for a 28.57-MHz (35-ns) device, and 20 megabits/s for a 40-MHz (25-ns) device. Since the serial port is fully static, it also functions at arbitrarily low clocking frequencies.

Two modes of operation are provided to support a wide range of applications:

❑ Continuous mode – provides operation that requires only one frame synchronization (frame sync) pulse to transmit several packets at maximum frequency

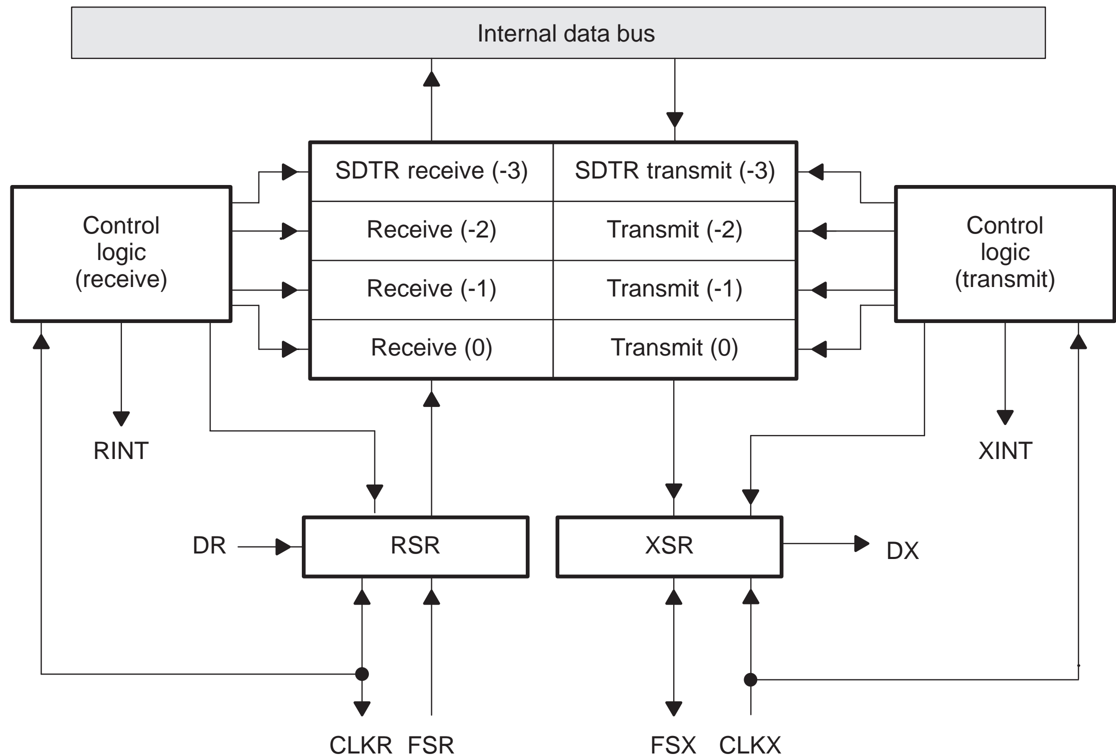
❑ Burst mode – allows transmission of a single 16-bit word following a frame sync pulse.

These two modes of operation suit most of the industry-standard synchronous serial-data devices, such as codecs. This port is intended to provide a glueless interface to most of the standard codec parts. However, these modes can also be adapted for specialized synchronous interfaces.

9.2 Components and Basic Operation

The synchronous serial port has several hard-wired parts, including two FIFO buffers and six signal pins. Figure 9–1 shows how the components of the synchronous serial port are interconnected.

Figure 9–1. Synchronous Serial Port Block Diagram



9.2.1 Signals

Serial port operation requires three basic signals:

❑ Clock signal. The clock signal (CLKX/CLKR) is used to control timing during the transfer. The timing signal for transmissions can be either generated internally or taken from an external source.

❑ Frame sync signal. The frame sync signal (FSX/FSR) is used at the start of a transfer to synchronize the transmit and receive operations. The frame sync signal for transmissions can be either generated internally or taken from an external source.

- Data signal. The data signal carries the actual data that is transferred in the transmit/receive operation. The data signal transmit pin (DX) of one device should be connected to the data signal receive (DR) pin on another device.

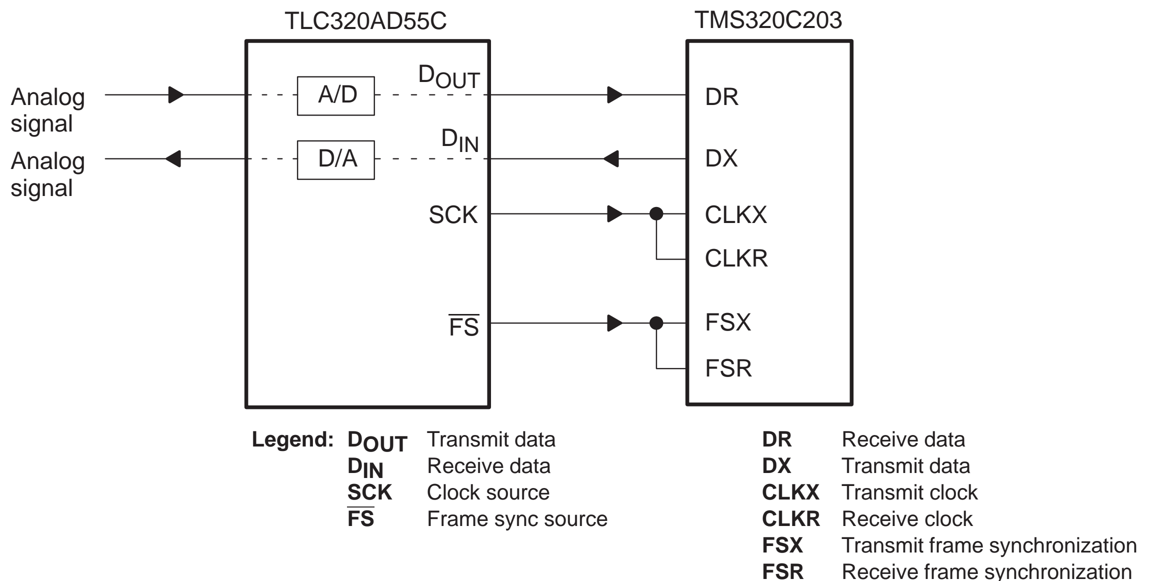
Table 9–1 describes the six pins that use these signals.

Table 9–1. SSP Interface Pins

Pin Name	Description
CLKX	*Transmit clock input or output.* The clock signal is used for clocking data from the serial port transmit shift register (XSR) to the DX pin. If the port is configured for accepting an external clock, this pin receives the clock signal. If the port is configured for generating an internal clock, this pin transmits the clock signal.
FSX	*Transmit frame synchronization.* FSX signals the start of a transmission. If the port is configured for accepting an external frame sync pulse, this pin receives the pulse. If the port is configured for generating an internal frame sync pulse, this pin transmits the signal.
DX	*Serial data transmit.* DX transmits serial data from the serial port transmit shift register (XSR).
CLKR	*Receive clock input.* CLKR receives an external clock signal for clocking the data from the DR pin into the serial port receive shift register (RSR).
FSR	*Receive frame synchronization.* FSR initiates the reception of data at the beginning of the packet.
DR	*Serial data receive.* DR receives serial data, transferring it into the serial port receive shift register (RSR).

Figure 9–2 shows how the signals are connected in a typical serial transfer between two devices. The DR pin receives serial data from the D_{OUT} signal, and the DX signal sends serial data to the D_{IN} pin. The FSX and FSR signals are both supplied from the $\overline{FS}$ pin, and they initiate the transfers (at the beginning of a data packet). The SCK signal drives both the CLKX and CLKR signals, which clock the bit transfers.

Figure 9–2. 2-Way Serial Port Transfer With External Frame Sync and External Clock



9.2.2 FIFO Buffers and Registers

The synchronous serial port (SSP) has two four-level transmit and receive FIFO buffers (shown at the center of Figure 9–1 on page 9-3).

Two on-chip registers allow you to access the FIFO buffers and control the operation of the port:

- **Synchronous data transmit and receive register (SDTR).** The SDTR, at I/O address FFF0h, is used for the top of both FIFO buffers (transmit and receive) and is the only visible part of the FIFO buffers.
- **Synchronous serial port control register (SSPCR).** The SSPCR, at I/O address FFF1h, contains bits for setting port modes, indicating the status of a data transfer, setting trigger conditions for interrupts, indicating error conditions, accepting bit input, and resetting the port. Section 9.3 includes a detailed description of the SSPCR.

Two other registers (not accessible to a programmer) control transfers between the FIFO buffers and the pins:

- **Synchronous serial port transmit shift register (XSR).** Each data word is transferred from the bottom level of the transmit FIFO buffer to the XSR. The XSR then shifts the data out (MSB first) through the DX pin.
- **Synchronous serial port receive shift register (RSR).** Each data word is accepted, one bit at a time, at the DR pin and shifted into the RSR. The RSR then transfers the word to the bottom level of the receive FIFO buffer.

9.2.3 Interrupts

The synchronous serial port (SSP) has two hardware interrupts that let the processor know when the FIFO buffers need to be serviced:

- ❑ Transmit interrupts (XINTs) cause a branch to address 000Ah in program space whenever the transmit-interrupt trigger condition is met. Set the trigger condition by setting bits FT1 and FT0 in the SSPCR (see Table 5–8 on page 5-26). XINTs have a priority level of 8 (1 being highest).
- ❑ Receive interrupts (RINTs) cause a branch to address 0008h in program space whenever the receive-interrupt-trigger condition is met. The trigger condition is selected by setting the FR1 and FR0 bits in the SSPCR (see Table 5–8 on page 5-26). RINTs have a priority level of 7.

These are maskable interrupts controlled by the interrupt mask register (IMR) and interrupt flag register (IFR).

Note:

To avoid a double interrupt from the SSP, clear the IFR bit (XINT or RINT) in the corresponding interrupt service routine, just before returning from the routine.

9.2.4 Basic Operation

Typically, transmission through the serial port follows this process:

1) Initialize the serial port to the desired configuration by writing to the SSPCR.
2) Your software writes up to four words to the transmit FIFO buffer through the SDTR.
3) The transmit FIFO buffer copies the earliest-written word to the transmit shift register (XSR) when the XSR is empty.
4) The XSR shifts the data, bit-by-bit (MSB first), to the DX pin.
5) When the XSR empties, it signals the FIFO buffer, and then:
   - ❑ If the FIFO buffer is not empty, the process repeats from step 3.
   - ❑ If the FIFO buffer is empty (as specified by the FT1 and FT0 bits in the SSPCR), it sends a transmit interrupt (XINT) to request more data, and the process repeats from step 2.

Reception through the serial port typically is done as follows:

1) Data from the DR pin is shifted, bit-by-bit (MSB first), into the receive shift register (RSR).

2) When the RSR is full, the RSR copies the data to the receive FIFO buffer.

3) The process then does one of two things, depending upon the state of the receive FIFO buffer:

   - If the receive FIFO buffer is not full, the process repeats from step 1.
   - If the receive FIFO buffer is full (as specified by the FR1 and FR0 bits in the SSPCR), it sends a receive interrupt (RINT) to the processor to request servicing.

4) The processor can read the received data from the receive FIFO buffer through the SDTR.

9.3 Controlling and Resetting the Port

The synchronous serial port control register (SSPCR) controls the operation of the synchronous serial port. To configure the serial port, a total of two writes to the SSPCR are necessary:

1) Write your choices to the configuration bits and place the port's FIFO in reset by writing zeros to SSPCR bits XRST and RRST.

2) Write your choices to the configuration bits and take the port's FIFO out of reset by writing ones to bits XRST and RRST.

Note:

XRST and RRST are bits that reset the pointer to two FIFOs (transmit and receive). These bits do not reset the serial port mode or operation. When XRST and RRST are reset, the FIFO pointers are set to start at zero (empty condition). See enhanced serial port features in section 9.8 to view the reset conditions in ESSP.

Set the DLB bit of the SSPCR to zero to disable digital loopback mode, which is not normally used in serial transfers. See section 9.7.1, *Test Bits*, for a description of digital loopback mode.

Make sure you write your configuration choices to the SSPCR during both writes.

Figure 9–3 shows the 16-bit memory-mapped SSPCR. Following the figure is a description of each of the bits.

Figure 9–3. Synchronous Serial Port Control Register (SSPCR) — I/O-Space FFF1h

15	14	13	12	11	10	9	8
FREE	SOFT	TCOMP	RFNE	FT1	FT0	FR1	FR0
R/W–0	R/W–0	R–0	R–0	R/W–0	R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
OVF	IN0	XRST	RRST	TXM	MCM	FSM	DLB
R–0	R–0	R/W–1	R/W–1	R/W–0	R/W–0	R/W–0	R/W–0

Note: R=Read access; W=Write access; value following dash (–) is value after reset.

Table 9–2. SSPCR — I/O-Space Address FFF1h Bit Descriptions

Bit No.	Name	Function
15–14	FREE, SOFT	These bits 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 breakpoint, the clock continues to run (that is, free runs) and data is shifted out. In this case, SOFT is a *don't care*. If FREE = 0, then SOFT takes effect. At reset, immediate stop mode is selected (FREE = 0 and SOFT = 0). The effects of the FREE and SOFT bits are: FREE SOFT Run/Emulation Mode 0 0 Immediate stop 0 1 Stop after completion of word 1 0 Free run 1 1 Free run Note: If an option besides immediate stop is chosen for the receiver, an overflow error is possible. The default mode (selected at reset) is immediate stop. The FREE and SOFT bits are for emulation and test purpose only. In your application, use '00' as default values for these bits.
13	TCOMP	Transmission complete. This bit is cleared to 0 when all data in the transmit FIFO buffer has been transmitted (the buffer is empty) and is set to 1 when new data is written to the transmit FIFO buffer (the buffer is not empty).
12	RFNE	Receive FIFO buffer not empty bit. This bit is 1 when the receive FIFO buffer contains data and is cleared when the buffer empties.
11–10	FT1, FT0	FIFO transmit-interrupt bits. The values you write to FT0 and FT1 set an interrupt trigger condition based on the contents of the transmit FIFO buffer. When this condition is met, a transmit interrupt (XINT) is generated and the data can be transferred out to the FIFO buffer using the OUT instruction. Writing to bits FT1 and FT0 controls transmit interrupt generation as follows: FT1 FT0 Generates XINT when... 0 0 Transmit FIFO buffer can accept one or more words; XINT occurs repeatedly until the buffer is full. 0 1 Transmit FIFO buffer can accept two or more words; XINT occurs repeatedly until three words are written. 1 0 Transmit FIFO buffer can accept three or four words; XINT occurs repeatedly until two words are written. 1 1 Transmit FIFO buffer is empty (can accept 4 words); XINT occurs repeatedly until one word is written.

Table 9–2. SSPCR — I/O-Space Address FFF1h Bit Descriptions (Continued)

Bit No.	Name	Function
9–8	FR1, FR0	FIFO receive-interrupt bits. The values you write to FR0 and FR1 set an interrupt trigger condition based on the contents of the receive FIFO buffer. When this condition is met, a receive interrupt (RINT) is generated and the data can be transferred in from the FIFO buffer using the IN instruction. Writing to bits FR1 and FR0 controls receive interrupt generation as follows: FR1 FR0 Generate RINT when... 0 0 Receive FIFO buffer is not empty. 0 1 Receive FIFO buffer holds at least two words. 1 0 Receive FIFO buffer holds at least three words. 1 1 Receive FIFO buffer is full (holds four words).
7	OVF	Overflow bit. This bit is set whenever the receive FIFO buffer is full and another word is received in the RSR. The contents of the FIFO buffer will not be overwritten by this new word. OVF is cleared when the FIFO buffer is read.
6	IN0	Input bit. This bit allows the CLKR pin to be used as a bit input. IN0 reflects the current logic level on the CLKR pin. IN0 can be tested by using a BIT or BITT instruction on the SSPCR. If the serial port is not used, IN0 can be used as a general-purpose bit input.
5	XRST	Transmit reset bit. This bit resets the transmitter FIFO of the serial interface. Set XRST to 0 to put the transmitter FIFO in reset. The FIFO will point to the start of the 4-deep FIFO and treat the FIFO as empty. Set XRST to 1 to bring the transmitter out of reset.
4	RRST	Receive reset bit. This bit resets the receiver FIFO of the serial interface. Set RRST to 0 to put the receiver FIFO in reset. The FIFO will point to the start of the 4-deep FIFO and treat the FIFO as empty. Set RRST to 1 to bring the receiver out of reset.
3	TXM	Transmit mode. This bit determines the source device for the frame synchronization (frame sync) pulse for transmissions. It configures the transmit frame sync pin (FSX) as an output or as in input. Note that the receive frame sync pin (FSR) is always configured as an input. 0 An external frame sync source is selected. FSX is configured as an input and accepts an external frame sync signal. The transmitter idles until a frame sync pulse is supplied on the FSX pin. 1 The internal frame sync source is selected. The FSX pin is configured as an output and sends a frame sync pulse at the beginning of every transmission. In this mode, frame sync pulses are generated internally when data is transferred from the SDTR to the XSR to initiate data transfers. The internally generated framing signal is synchronous with respect to CLKX.

Table 9–2. SSPCR — I/O-Space Address FFF1h Bit Descriptions (Continued)

Bit No.	Name	Function	
2	MCM	Clock mode. This bit determines the source device for the clock for a serial port transfer. It configures the clock transmit pin (CLKX) as an output or as an input. Note that the clock receive pin (CLKR) is always configured as an input.	
		0	An external clock source is selected. The CLKX pin is configured as an input that accepts an external clock signal.
		1	The internal clock source is selected. The CLKX pin is configured as an output driven by an internal clock source with a frequency equal to 1/2 that of CLKOUT1. Note that if MCM = 1 and DLB = 1, CLKR is also supplied by the internal source.
1	FSM	Frame synchronization mode. The FSM bit specifies whether frame synchronization pulses are required between consecutive word transfers.	
		0	Continuous mode is selected. In continuous mode, one frame sync pulse (FSX/FSR) initiates the transmission/reception of multiple words.
		1	Burst mode is selected. A frame sync pulse (FSX/FSR) is required for the transmission/reception of each word.
0	DLB	Digital loopback mode. The DLB bit can be used to put the serial port in digital loopback mode.	
		0	Digital loopback mode is disabled. The DR, FSR, and CLKR signals are connected to their respective device pins.
		1	Digital loopback mode is enabled. DR and FSR become internally connected to DX and FSX, respectively. The FSX and DX signals appear on the device pins, but FSR and DR do not. TXM must be set to 1 for proper operation in digital loopback mode. CLKX drives CLKR if you also set 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.

9.3.1 Selecting a Mode of Operation (Bit 1 of the SSPCR)

Different applications require different modes of operation for the serial port. The synchronous serial port supports two basic modes of operation:

- **Continuous mode (FSM = 0).** The continuous mode of operation requires only an initial frame sync pulse, as long as a write to SDTR (for transmission) or a read from SDTR (for reception) is executed during each transmission/reception. Use continuous mode for transmitting a continuous stream of information.

- **Burst mode (FSM = 1).** In burst mode operation, a frame sync is required for every transfer, and there are periods of serial port inactivity between packet transmits. Use this mode for transmitting short packets of information.

9.3.2 Selecting Transmit Clock Source and Transmit Frame Sync Source (Bits 2 and 3 of the SSPCR)

The transmit clock is used to set the transmission rate of the serial port. Transmissions can be clocked by the internal clock source or by an external source:

- To use the *internal clock source*, set the MCM bit in the SSPCR to 1. This causes the serial port to take CLKX from the internal source. The internal clock rate is (CLKOUT1 rate)/2.

- To use an *external clock source:*
  1) Connect the external clock to the CLKX pin of the transmitter and to the CLKR pin of the receiver.
  2) Set the MCM bit to 0 in the SSPCR to cause the serial port to get CLKX from the CLKX pin.

A transmit frame sync pulse marks the start of a data transmission. The synchronous serial port can transmit using the internal frame sync source or using an external source:

- To use *internal frame sync pulses*, set the TXM bit in the SSPCR to 1.

- To use *external frame sync pulses:*
  1) Connect the frame sync source to the FSX pin of the transmitter and to the FSR pin of the receiver.
  2) Set the TXM bit in the SSPCR to 0 to enable external frame syncs.

The source configuration options are summarized in Table 9–3.

Table 9–3. Selecting Transmit Clock and Frame Sync Sources

MCM	TXM	CLKX source	FSX source
0	0	External	External
0	1	External	Internal
1	0	Internal	External
1	1	Internal	Internal

9.3.3 Resetting the Synchronous Serial Port (Bits 4 and 5 of the SSPCR)

Reset the synchronous serial port by setting XRST = 0 and RRST = 0 and then setting XRST = 1 and RRST = 1. These bits can be set individually, allowing you to reset only the transmitter or only the receiver. When a zero is written to one of these bits, activity in the corresponding section of the serial port stops.

9.3.4 Using Transmit and Receive Interrupts (Bits 8–11 of the SSPCR)

The synchronous serial port has two interrupts for managing reads and writes to the FIFO buffers. The processor can determine when the FIFO buffers need servicing in two ways:

- By polling the SSPCR register (RFNE and TCOMP bits)
- By setting up XINT and/or RINT interrupts

To determine when the FIFO buffers need servicing by polling, disable the interrupts by masking them in the interrupt mask register (IMR).

If you want to use interrupts to manage your serial transfer, then perform three steps:

1) Create interrupt service routines for XINTs and RINTs and include a branch to each service routine at the appropriate interrupt vector address:

   - The RINT vector is fetched from address 0008h.
   - The XINT vector is fetched from address 000Ah.

2) Select when you want interrupts to occur and set the FR0, FR1, FT0, and FT1 bits accordingly. You can set the FIFO buffers to generate interrupts when they are empty, when they have 1 or 2 words, when they have 3 or 4 words, or when they are full. Table 5–8 shows what values to set in the FR0, FR1, FT0, and FT1 bits for each condition.

3) Enable the interrupts by unmasking them in the interrupt mask register (IMR).

For more information about interrupts, see section 5.6, *Interrupts*, p. 5-15.

Note:

To avoid a double interrupt from the SSP, clear the IFR bit (XINT or RINT) in the corresponding interrupt service routine, just before returning from the routine.

9.4 Managing the Contents of the FIFO Buffers

The SDTR is a read/write register (at I/O address FFF0h) that is used to send data to the transmit FIFO buffer and to extract data from the receive FIFO buffer.

A word is written to the SDTR by the OUT instruction. When the transmit FIFO buffer is full, additional writes to the SDTR are ignored. Therefore, your program should not write a word for transmission until at least one space is available in the transmit FIFO buffer. You can set up a transmit interrupt (XINT) based on the contents of the buffer (using the FT1 and FT0 bits of the SSPCR). If your program writes words to the buffer only when the buffer is empty, you can use the transmission complete (TCOMP) bit; when the buffer is empty, TCOMP = 0.

When the receive FIFO buffer holds data, you can read the received data from the FIFO buffer through the SDTR (using the IN instruction). You can check the state of the receive buffer by reading the receive FIFO buffer not empty (RFNE) bit in the SSPCR, or you can set up a receive interrupt (RINT) based on the state of the buffer (using the FR1 and FR0 bits of the SSPCR).

9.5 Transmitter Operation

Transmitter operation is different in continuous and burst modes. Other differences also depend on whether an internal or an external frame sync is used.

9.5.1 Burst Mode Transmission With Internal Frame Sync (FSM = 1, TXM = 1)

Use burst mode transmission with internal frame sync to transfer short packets at rates lower than maximum packet frequency while using an internal frame sync generator. Place the transmitter in burst mode with internal frame sync by setting the FSM bit to 1 and the TXM bit to 1.

This mode of operation offers several features:

- A one-clock-cycle frame-sync pulse is generated internally at the beginning of each transmission.
- Continuous transmission is possible if SDTR is updated in the XINT interrupt service routine.
- Transmission can be initiated by an external event (for example, an external interrupt) or by a receive interrupt (RINT).

Generally, the transmit clock and the receive clock have the same source. This allows each bit to be transmitted from another device on a rising edge of the clock signal and received by the ’C20x on the next falling edge of the clock signal.

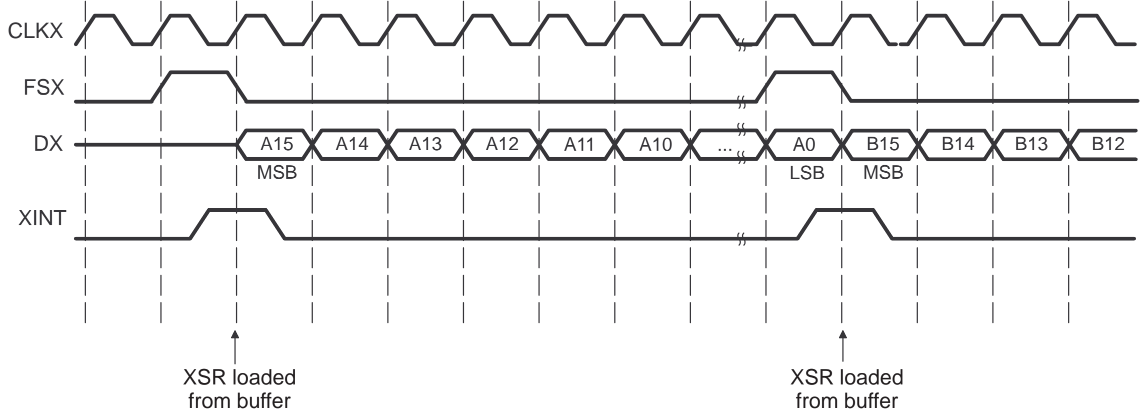
Burst mode transmission with internal frame sync requires the following order of events (see Figure 9–4):

1) Initiate the transfer by writing to SDTR.
2) A frame sync pulse is generated on the next rising edge of CLKX. The frame sync pulse remains high for one clock cycle.
3) On the next rising edge of CLKX after FSX goes high, XSR is loaded with the value at the bottom of the FIFO buffer, and the frame sync pulse goes low. Additionally, the first data bit (MSB first) is driven on the DX pin. If the FIFO buffer becomes empty during this operation, it generates XINT to request more data.
4) The rest of the bits are then shifted out. Each new bit is transmitted at each consecutive rising edge of CLKX.
5) If the FIFO buffer still holds a word or words to be transmitted, another frame sync pulse is generated in parallel to the driving of the LSB on the DX pin, and transmission continues at step 3. If the FIFO is empty, transmission is complete.

If the SDTR is loaded with a new word while the transmit FIFO buffer is full, the new word will be lost; the FIFO buffer will not accept any more than four words.

The burst mode can be discontinued (changed to continuous mode) only by a serial-port or device reset. Changing the FSM bit during transmit or halt will not necessarily cause a switch to continuous mode.

Figure 9–4. Burst Mode Transmission With Internal Frame Sync and Multiple Words in the Buffer



9.5.2 Burst Mode Transmission With External Frame Sync (FSM = 1, TXM = 0)

Use burst mode transmission with external frame sync to transfer short packets at rates lower than maximum packet frequency while using an external frame sync generator. Place the transmitter in burst mode with external frame sync by setting the FSM bit to 1 and the TXM bit to 0.

This mode of operation offers several features:

- A frame sync pulse initiates transmission.
- If a frame sync pulse occurs after the initial one, then transmission restarts.
- Transmission can be initiated by an external event (for example, an external interrupt) or by a serial port receive interrupt (RINT).

Generally, the transmit clock and the receive clock have the same source. This allows each bit to be transmitted from another device on a rising edge of the clock signal and received by the ’C20x on the next falling edge of the clock signal.

Burst mode transmission with external frame sync involves the following order of events (see Figure 9–5):

1) A frame sync pulse initiates the transmission. The pulse is sampled on the falling edge of CLKX. After the falling edge of CLKX, the contents of the first entry in the FIFO buffer are transferred to the XSR. If the FIFO buffer becomes empty during this operation, it generates a XINT to request more data.

2) On the next rising edge of CLKX after FSX goes high, DX is driven with the first bit (MSB) of the word to be transmitted.

3) The frame sync goes low (and remains low during word transmission).

4) Once FSX goes low, the rest of the bits are shifted out.

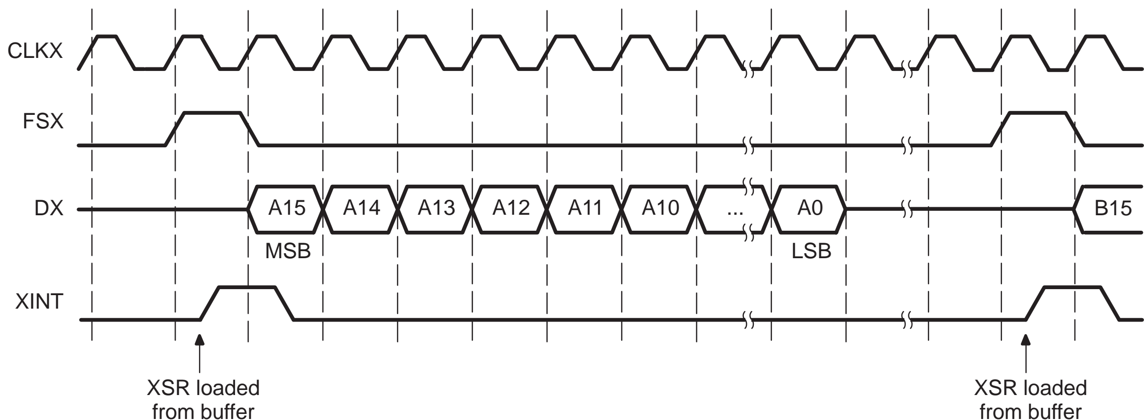
5) When all of the bits in the word are transferred, the port waits for a new frame sync pulse.

If the SDTR is loaded with a new word while the transmit FIFO buffer is full, the new word will be lost; the FIFO buffer will not accept any more than four words.

If a frame sync pulse occurs during transmission, transmission is restarted. If another value has been written to the SDTR, a new word is sent; otherwise, the last word in the XSR is sent.

The burst mode can be discontinued (changed to continuous mode) only by a serial-port or device reset. Changing the FSM bit during transmit or halt will not necessarily cause a switch to continuous mode.

Figure 9–5. Burst Mode Transmission With External Frame Sync



9.5.3 Continuous Mode Transmission With Internal Frame Sync (FSM = 0, TXM = 1)

Use continuous mode transmission with internal frame sync to transfer long packets at maximum packet frequency while using an internal frame sync generator. Place the transmitter in continuous mode with internal frame sync by setting the FSM bit to 0 and the TXM bit to 1.

In continuous mode, frame sync pulses are not necessary after the initial pulse for consecutive packet transfers. A frame sync is generated only for the first transmission. As long as the FIFO buffer has new values to transmit, the mode continues. Transmission halts when the buffer empties. If SDTR is written to after the halt, the device starts a new continuous mode transmission.

This mode of operation offers several features:

- A write to the SDTR begins the transmission.
- A one-clock-cycle frame-sync pulse is generated internally at the beginning of the transmission.
- As long as data is maintained in the transmit FIFO buffer, the mode continues.
- Failure to update the FIFO buffer causes the process to end.

Generally, the transmit clock and the receive clock have the same source. This allows each bit to be transmitted from another device on a rising edge of the clock signal and received by the ’C20x on the next falling edge of the clock signal.

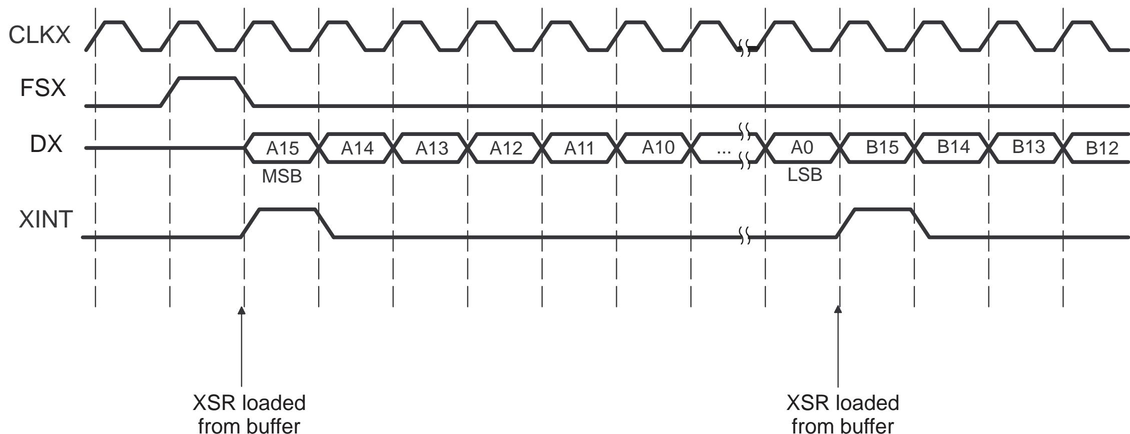
As illustrated by Figure 9–6, in this mode, the port operates as follows:

1) The transfer is initiated by a write to the SDTR.
2) The write to the SDTR causes a frame sync pulse to be generated on the next rising edge of CLKX. The frame sync pulse remains high for one clock cycle.
3) On the next rising edge of CLKX after FSX goes high, the XSR is loaded with the earliest-written value from the transmit FIFO buffer, and the frame sync pulse goes low. Additionally, the first data bit (MSB first) is driven on the DX pin. If the FIFO buffer becomes empty during this operation, it generates an XINT to request more data.
4) The rest of the bits are then shifted out. Each new bit is transmitted at the rising edge of CLKX.
5) Once the entire word in the XSR is shifted out, the next word is loaded in and the first bit of the word is placed on the DX pin. Then, the process repeats beginning with step four. If a new word is not in the transmit FIFO buffer, the process ends.

If the SDTR is loaded with a new word while the transmit FIFO buffer is full, the new word will be lost; the FIFO buffer will not accept any more than four words.

Continuous mode can be discontinued (changed to burst mode) only by a serial-port mode change or device reset. Changing the FSM bit during transmit or halt will not necessarily cause a switch to burst mode.

Figure 9–6. Continuous Mode Transmission With Internal Frame Sync



9.5.4 Continuous Mode Transmission with External Frame Sync (FSM=0, TXM=0)

Use continuous mode transmission with external frame sync to transfer long packets at maximum packet frequency while using an external frame sync generator. Place the transmitter in continuous mode with external frame sync by setting the FSM bit to 0 and the TXM bit to 0.

In continuous mode, frame sync pulses are not necessary after the initial pulse for consecutive packet transfers. A frame sync is generated only for the first transmission. As long as the FIFO buffer has new values to transmit, the mode continues. Transmission halts when the buffer empties. If SDTR is written to after the halt, the device starts a new continuous mode transmission.

This mode of operation offers several features:

- ❑ Only one frame sync is necessary for the transmission of consecutive packets.
- ❑ If the FIFO buffer is not empty, the mode continues. If the FIFO buffer is empty, the process ends.

Generally, the transmit clock and the receive clock have the same source. This allows each bit to be transmitted from another device on a rising edge of the

clock signal and received by the ’C20x on the next falling edge of the clock signal.

Continuous mode transmission with external frame sync requires the following order of events (see Figure 9–7):

1) A frame sync pulse initiates the transmission. The pulse is sampled on the falling edge of CLKX. After the falling edge of CLKX, the contents of the current word in the transmit FIFO buffer are transferred to the XSR. If the FIFO buffer becomes empty during this operation, it generates an XINT to request more data.

2) On the next rising edge of CLKX after FSX goes high, DX is driven with the first bit (MSB) of the word to be transmitted.

3) The frame sync goes low (and remains low during word transmission).

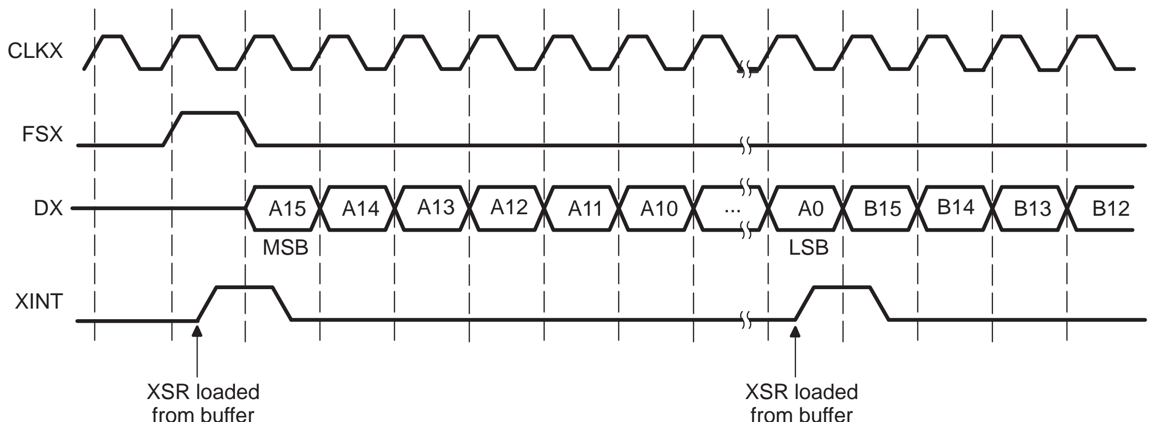
4) Once FSX goes low, the rest of the bits are shifted out.

5) Once the entire word in the XSR is shifted out, the next word is loaded in and the first bit of the word is placed on the DX pin. Then, the process repeats beginning with step four. If a new word is not in the transmit FIFO buffer, the process ends.

If the SDTR is loaded with a new word while the transmit FIFO buffer is full, the new word will be lost; the FIFO buffer will not accept any more than four words.

The continuous mode can be discontinued (changed to burst mode) only by a serial-port or device reset. Changing the FSM bit during transmit or halt will not necessarily cause a switch to burst mode.

Figure 9–7. Continuous Mode Transmission With External Frame Sync



9.6 Receiver Operation

Receiver operation is different in continuous and burst modes. The receiver does not generate frame sync pulses; it always takes the frame sync pulse as an input.

In selecting the proper receive mode, note that the mode for the receiver must match the mode for the transmitter.

If all four words of the receive FIFO buffer have been filled, the buffer will not accept additional words. If a fifth write is attempted, the overflow (OVF) bit of the SSP control register (SSPCR) is set to 1.

9.6.1 Burst Mode Reception

Use burst mode receive to transfer short packets at rates lower than maximum packet frequency.

This mode of operation offers these features:

- The data packet is marked by the frame sync pulse on FSR.
- Reception of data can be maintained continuously.

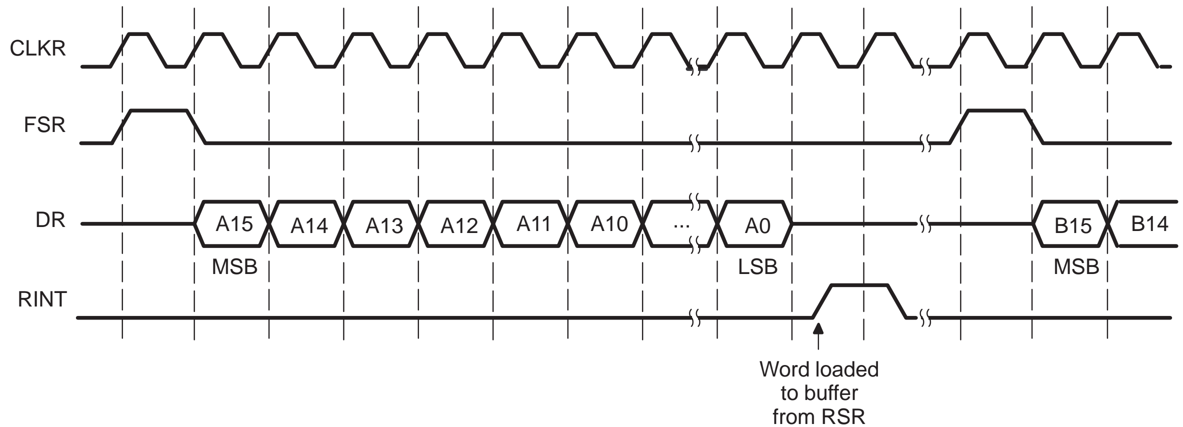
Generally, the transmit clock and the receive clock have the same source. This allows each bit to be transmitted from another device on a rising edge of the clock signal and received by the ’C20x on the next falling edge of the clock signal.

The following events occur during a burst mode receive operation (see Figure 9–8):

1) A frame sync pulse initiates the receive operation. This event is sampled on the falling edge of CLKR.
2) On the next falling edge of CLKR after the falling edge of FSR, the first bit (MSB) is shifted into the receive shift register (RSR).
3) The rest of the bits in the word are then shifted into RSR one at a time at each consecutive falling edge of CLKR.
4) After all bits have been received, if the receive FIFO buffer is not full, the contents of the RSR are copied into the receive FIFO buffer. If the FIFO buffer becomes full during this operation, an interrupt (RINT) is sent to the CPU, and the overflow bit (OVF) of the SSPCR is set.
5) The receive operation is started again after the next frame sync pulse. However, the received word can be loaded into the FIFO buffer only if the buffer is empty; otherwise, the word is lost.

If a frame sync pulse occurs during reception, reception is restarted, and the bits that were shifted into the RSR before the pulse are lost.

Figure 9–8. Burst Mode Reception



9.6.2 Continuous Mode Reception

Use continuous mode receive to transfer long packets at maximum packet frequency.

This mode of operation offers several features:

- ❑ Only the first frame sync signal is necessary to start the reception of consecutive words.
- ❑ As long as the receive FIFO buffer is not allowed to overflow, the mode continues. Overflow is indicated by the OVF bit in the SSPCR.
- ❑ Reception can be maintained continuously.

Generally, the transmit clock and the receive clock have the same source. This allows each bit to be transmitted from another device on a rising edge of the clock signal and received by the ’C20x on the next falling edge of the clock signal.

As shown in Figure 9–9, the following events occur during a continuous mode receive operation:

1) The receive operation begins when a frame sync signal is detected on the falling edge of CLKR.

2) On the first falling edge of CLKR after the frame sync signal goes low, the first bit (MSB) is shifted into the RSR.

3) The remaining bits in the word are then shifted into the RSR, one by one at the falling edge of each consecutive clock cycle.

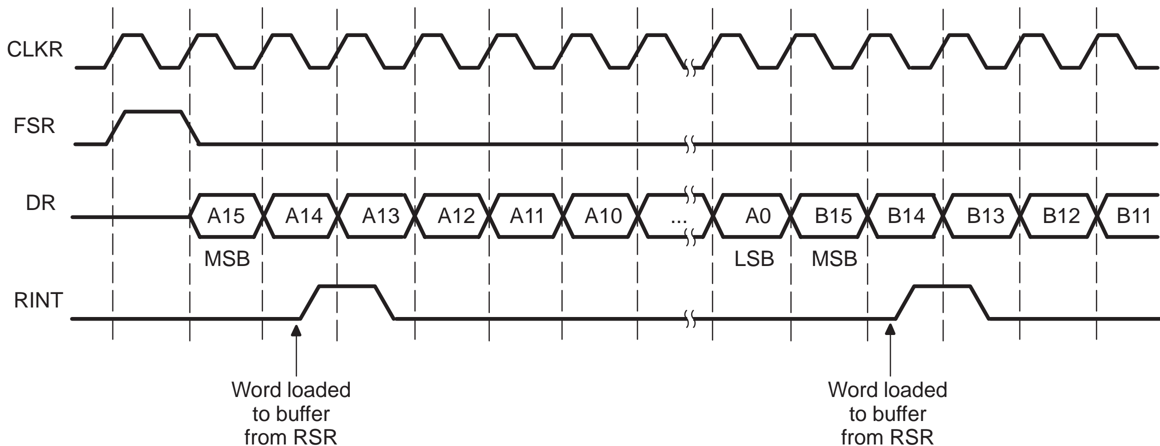
4) After all bits have been received, if the FIFO buffer is not full, the contents of the RSR are copied to the receive FIFO buffer. If the receive FIFO buffer does become full, an interrupt (RINT) is sent to the CPU, and if overflow has occurred, the overflow (OVF) bit of the SSPCR is set.

5) The process then repeats itself, except that there are no additional frame sync pulses.

If a frame sync pulse occurs during reception, then reception is restarted and the bits in the current word that were shifted into the RSR before the pulse are lost.

If the FIFO buffer becomes full, no new words will be received into the buffer until at least one word has been read from the buffer (through the SDTR). Once the continuous reception is started, the port will always be reading in the values on the DR pin. To stop continuous mode reception, either change mode bits to burst mode or initiate system reset.

Figure 9–9. Continuous Mode Reception



9.7 Troubleshooting

The synchronous serial port uses three bits for troubleshooting and testing. In addition to using these three bits, you must be able to identify special error conditions that may occur in actual transfers. Error conditions result from an unprogrammed event occurring to the serial port. These conditions are operational errors such as overflow, underflow, or a frame sync pulse during a data transfer.

This section describes how the serial port handles these errors and the state it acquires during these error conditions. The types of errors differ slightly in burst and continuous modes.

9.7.1 Test Bits

Three bits in the SSPCR help you test the synchronous serial port. The digital loopback mode bit (DLB) can be used to internally connect the receive data and frame sync signals to the transmit data and frame sync signals on the same device. The FREE and SOFT bits allow emulation modes that stop the port either immediately or after the transmission of the current word. Figure 9–10 shows the bits that are used for troubleshooting. The list items following the figure describe the functions of these bits.

Figure 9–10. Test Bits in the SSPCR

- **FREE and SOFT** are special emulation bits that allow you to 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 (that is, free runs) and data is shifted out. In this case, SOFT is a *don't care*. But if FREE is 0, then SOFT takes effect. If SOFT = 0, then the clock immediately stops, thus aborting any transmission. If the SOFT bit is 1, the particular transmission continues until completion of the word, and then the clock halts. Table 9–4 summarizes the available run and emulation modes.

Table 9–4. Run and Emulation Modes

FREE	SOFT	Run/Emulation Mode
0	0	Immediate stop
0	1	Stop after completion of word
1	0	Free run
1	1	Free run

Note:

If an option besides immediate stop is chosen for the receiver, an overflow error is possible. The default mode (selected at reset) is immediate stop.

❑ **DLB** enables or disables digital loopback mode:

- To enable the digital loopback mode, set DLB = 1.
- To disable the digital loopback mode, set DLB = 0.

When you enable digital loopback mode, the transmit data (DX) and frame sync (FSX) signals become internally connected to the receive data (DR) and frame sync (FSR) signals. After writing code for both the transmitter and the receiver, you can then test whether the code is working properly and also check that the serial port is functioning. In addition, if both the DLB and MCM bits are 1, the transmit clock signal is also connected internally to the receive clock signal.

The serial port operates normally when you disable digital loopback mode; that is, no transmit and receive signals are internally connected together.

Note:

To configure the serial port, a total of two writes to the SSPCR are necessary:

1) First, write your choices to the configuration bits and place the port in reset by writing zeros to XRST and RRST.

2) Second, write your choices to the configuration bits and take the port out of reset by writing ones to the XRST and RRST bits.

9.7.2 Burst Mode Error Conditions

The following are descriptions of errors that can occur in burst mode:

- Underflow. Underflow is caused if an external FSX occurs, and there are no new words in the transmit FIFO buffer. Upon receiving the FSX (generally, from an external clock source), transmitter resends the previous word; that is, the value in XSR will be transmitted again.
- Overflow. This error occurs when the device has not read incoming data and more data is being sent (indicated by a frame sync pulse on FSR). The OVF bit of the SSPCR is set to indicate overflow. The processor halts updates to the FIFO buffer until the SDTR is read. Thus, any further data sent is lost.
- Frame sync pulse during a reception. If the frame sync occurs during a reception, the present reception is aborted and a new one begins. The data that was being loaded into the RSR is lost, but the data in the FIFO buffer is not. No RSR-to-FIFO buffer copy occurs until all 16 bits in a word have been received.
- Frame sync pulse during a transmission. Another error results when a frame sync occurs while a transmission is in process. If the data in the XSR is being driven on the DX pin when the frame sync pulse occurs, then the present transmission is aborted. Then, whatever data is next in the FIFO buffer at the time of the frame sync pulse is transferred to XSR for transmission.

9.7.3 Continuous Mode Error Conditions

The following are descriptions of continuous mode errors and how the port responds to them:

- Underflow. Underflow occurs when the XSR is ready to accept new data but there are no new words in the transmit FIFO buffer. Underflow errors are fatal to a transmission; it causes transmission to halt. For as long as the transmit FIFO buffer is empty, frame sync pulses are ignored. If new data is then written to the SDTR, another frame sync pulse is required (or generated, if you are using internal frame syncs) to restart continuous mode transmission.

  Your software can do the following to determine how many words are left in the transmit FIFO buffer:

  - Test for the condition TCOMP = 0. When the transmit FIFO buffer empties, the TCOMP bit of the SSPCR is set to 0.
  - Cause an interrupt (XINT) to occur based on the contents of the buffer. You can use bits FT1 and FT0 in the SSPCR to set the interrupt trigger conditions shown in Table 5–8 on page 5-26.

- Overflow**.** Overflow occurs when the RSR has new data to pass to the receive FIFO buffer but the FIFO buffer is full. Overflow errors are fatal to a reception. For as long as the FIFO buffer is full, any incoming words will be lost. To restart reception, make space in the buffer by reading from it (through the SDTR).

- Frame sync pulse during a transmission. After the initial frame sync, no others should occur during transmission. If a frame sync pulse occurs during a transmission, the current transmission is aborted, and a new transmit cycle begins.

- Frame sync pulse during a reception. After the initial frame sync, no others should occur during reception. If a frame sync pulse occurs during a reception, the current packet of data is lost. On any FSR pulse, the RSR bit counter is reset; therefore, the data that was being shifted into the RSR from the the DR pin is lost.

9.8 Enhanced Synchronous Serial Port (ESSP)

The enhanced synchronous serial port (ESSP) is a feature available in TMS320F206 and TMS320C206/LC206 series of digital signal processors. The ESSP is an enhancement of the synchronous serial port (SSP), which is standard in the C20x family. In addition to providing a glueless interface for multiple serial devices, the ESSP also features a pseudo serial peripheral interface (SPI) mode of operation. The maximum transmission rate for both transmit and receive operations are the CPU clock divided by two, i.e. CLKOUT1(frequency)/2. Therefore, the maximum rate is 10Mbit/s at 50ns, 14.28Mbit/s at 35ns, and 20Mbit/s at 25ns. Refer to the TI web site at **www.ti.com** and follow the *DSP* path to *'C20x DSP* to find software source on ESSP test programs.

9.8.1 ESSP Features

- Full-duplex, double-buffered synchronous serial port
- Highly flexible operation:
  - Burst and continuous modes
  - Supports 8- and 16-bit word lengths
  - Multichannel mode with glueless interface to as many as four voice-band or telephony codecs for telecommunications applications such as line cards and feature phones.
  - Pseudo serial peripheral interface (SPI) mode
- Independent four-level deep FIFO for both the receive and transmit sections
  - Programmable FIFO level interrupts to reduce software overhead
  - FIFO level status bits
- Various clocking options to ease interfacing in many applications
  - Internal shift clock, CLKX, derived from an independent 8-bit prescaler
  - Internal frame sync, FSX, derived from an independent 8-bit prescaler
  - Polarity control on shift clock, CLKX, and frame sync pulse, FSX
- High impedance control on data transmit pin DX for TDM applications
- Prescalers are configurable as general-purpose 16-bit counters.
- Fast transfer rate of 20 Mbits/s at 25ns cycle time

9.9 ESSP Pins

The enhanced synchronous serial port has seven pins for external interface. Table 9–5 explains the functions of these pins. In this table, SSP mode indicates that only one serial device is connected to the DSP chip (for example, the ESSP mode has not been activated. ESSP mode indicates that the ESSP features have been activated (by programming the ESSP registers) and that one or more serial devices have been connected to the DSP chip.

Table 9–5. TMS320C20x Enhanced Synchronous Serial Port Interface Signals

100 Pin	'C20x Pin	I/O/Z†	Description
87	CLKX	I/O	Transmit clock (input or output). Clock signal for clocking data from the serial port transmit shift register (XSR) to the data transmit (DX) pin. CLKX is an input if the MCM bit in the SSPCR is set to 0 (external CLKX). It can also be generated internally if the MCM bit is set to 1. Internal CLKX rate is determined by the input clock to the CLKX prescaler (CLXCT) and is governed by the equation: CLKX rate = CLKOUT1 / (2*(CLXCT+1)) The generated CLKX can also feed a frame sync prescaler (FSXCT) to generate internal frame syncs synchronous to CLKX at variable rates. The prescalers for CLKX and FSX are defined in the I/O register SSPCT at FFF3h in I/O space. The input to the CLKX prescaler is CLKOUT1.
84	CLKR/ FSX2	I/O	Receive clock (input). In the SSP mode, this pin is the external clock signal for clocking data from the DR (data receive) pin into the RSR (receive shift register) and must be present during serial port data receive process. If the serial port is not being used, this pin can be sampled as an input via the IN0 bit of the SSPCR. Frame synchronization pulse 2 (output). In the ESSP mode, if the multichannel register is configured for two channels, this pin transmits the frame sync for the second serial device connected to the serial port.
85	FSR/FSX3	I/O	Frame synchronization pulse for receive (input). In the SSP mode, the falling edge of the FSR pulse initiates the data receive process. Frame synchronization pulse 3 (output). In the ESSP mode, if the multichannel register is configured for three channels, this pin transmits the frame sync for the third serial device connected to the serial port.
86	DR	I	Serial data receive (input). Serial data is received into the receive shift register (RSR) from DR pin.

† I = Input, O = Output, Z = High impedance

Table 9–5. TMS320C20x Enhanced Synchronous Serial Port Interface Signals (Continued)

100 Pin	'C20x Pin	I/O/Z†	Description
89	FSX/FSX1	I/O	Frame synchronization pulse for transmit (input or output). The falling edge of the FSX pulse initiates the data transmit process beginning the clocking of the XSR. Following reset, FSX is an input. This pin can be selected by software to be an output when the TXM bit in the SSPCR is set to 1. The frame sync can be generated internally. The frame sync rate can be either defined by the prescaler FSXCT or by the rate at which data is written into the transmit FIFO. The internal CLKX can also feed a frame sync prescaler to generate internal frame sync synchronous to CLKX and at variable rates. Internal FSX rate is determined by the input clock to the prescaler and is governed by the equation: FSX rate = CLKX pin clock /((2*(FSXCT+1)) The prescalers for CLKX and FSX are defined in the I/O register SSPCT at FFF3h in I/O space. Frame synchronization pulse 1 (output). In the ESSP mode, this pin transmits the frame sync for the first serial device connected to the serial port. This frame sync functions as the master frame sync, while FSX2, FSX3, FSX4 follow this pulse as slaves.
90	DX	O	Serial data transmit (output). Serial data is transmitted from the transmit shift register (XSR) through DX pin. DX is placed in high impedance when not transmitting.
96	IO0/FSX4	I/O	Input/Output 0 (input or output). In the SSP mode, this pin is used as a general-purpose input/output. Frame synchronization pulse 4 (output). In the ESSP mode, if the multi-channel register is configured for four channels, this pin transmits the frame sync for the fourth serial device connected to the serial port.

† I = Input, O = Output, Z = High impedance

9.9.1 Multichannel Mode

In the multichannel mode of the ESSP, up to four serial devices can be connected gluelessly to the DSP. All the four serial devices are connected in parallel to the DX, DR, CLKX lines. In effect, all the serial devices transmit and receive data at the same shift clock rate. The exact instant at which each device transmits and receives data is determined by the frame sync pulse for the corresponding device. In the SSP mode, only one device is connected to the DSP and the default frame sync signal FSX is used. When additional serial devices are connected in the ESSP mode, CLKR, FSR and IO0 act as the frame syncs for the additional serial channels. The successive frame syncs are separated by 18 shift clocks.

9.10 ESSP Registers

The enhanced synchronous serial port operates through the five registers (SDTR, SSPCR, SSPST, SSPMC, and SSPCT) that are mapped into the I/O space. Before the ESSP can be used, the control and status registers need to be programmed. The ESSP registers are listed in Table 9–6.

Table 9–6. ESSP Registers

Registers	I/O Address	Value at Reset	Description
SSPST	FFF2h	0000h	SSP Status register
SSPMC	FFF3h	0000h	SSP Multichannel register
SSPCT–CLXCT	FFFBh	xx00h	Shift clock prescaler (CLKX) (low byte, bits 7–0)
SSPCT–FSXCT	FFFBh	00xxh	Frame sync prescaler (FSX) (high byte, bits 15–8)

Notes: 1) x – Indicates undefined values or value based on the pin levels at reset.
2) SSPST, SSPMC and SSPCT are registers that are unique to ESSP.

9.10.1 Synchronous Serial Port Status Register (SSPST)

The SSPST register is used to configure the various ESSP options. It has additional FIFO status bits. The prescalers for CLKX and FSX are also configured by the SSPST.

Figure 9–11. Synchronous Serial Port Status (SSPST) Register — I/O address FFF2h

15	14	13	12	11	10	9
DRP Pin	FSN	FSXOX	FSXST Status	Reserved	CLN	CLXOX
R	R/W	R/W	W1C/R		R/W	R/W

8	7–5	4–2	1	0
PRSEN	Transmit FIFO Status	Receive FIFO Status	SGNEX (Sign-Extend)	BYTE (8/16 Bit)
R/W	R	R	R/W	R/W

Note: R = Read, W = Write, W1C/R = Write one to clear

Table 9–7. SSPST Register — I/O address FFF2h Bit Descriptions

Bit No.	Name	Function
15	DRP pin	DR pin read bit. Read-only DRP bit that gives visibility to the DR pin.
14	FSN	Frame sync invert bit. FSN selects the polarity for the frame sync. At reset, FSN is 0 and selects FSX to be high for one CLKX duration. The data transmit and receive is based on the falling edge of FSX. If FSN is set to 1, the polarity of the FSX is inverted. The FSX remains high during data transmit or receive (8/16 CLKX cycles). FSN bit controls both the FSX and FSR polarity. In the internal FSX mode, the outgoing FSX is inverted once and the incoming FSR is inverted once. Thus, if FSX and FSR pins are externally connected, the polarity of the FSX/FSR are the same with respect to the SSP core.
13	FSXOX	Internal FSX selection bit. FSXOX selects the type of internal frame sync that is issued from the FSX pin. If set to 1, the FSX is from the frame sync prescaler FSXCT. If reset to 0, the internal FSX is at the rate at which data is written into the transmit FIFO.
12	FSXST Status	Prescaler FSXST status bit. FSXST is set to 1 every time the FSXCT prescaler counter reaches zero. FSXST can be read and cleared by writing a 1. This bit is also a counter-status bit in the 16-bit counter mode. It is set to 1 whenever the 16-bit counter reaches zero. FSXST initiates an interrupt if GPI is enabled in the SSPMC register.
11	Reserved	Reserved
10	CLN	Shift clock CLKX invert bit. CLN selects the polarity for the shift clock CLKX. If reset to 0, CLKX is of normal polarity. If set to 1, CLKX is inverted for internal and external CLKX. CLN bit controls both the CLKX and CLKR polarity. In the internal CLKX mode, the outgoing CLKX is inverted once, and the incoming CLKR signal is inverted once. Thus, if CLKX and CLKR pins are externally connected, the polarity of the CLKX/CLKR are the same with respect to the SSP core.
9	CLXOX	Input clock source CLXOX bit. In the general purpose counter mode (GPC bit =1), CLXOX selects the input clock source to the 16-bit counter (SSPCT). If CLXOX = 1, the input clock is CLKX pin clock (either CLKOUT1/2 or external CLKX depending on the MCM bit). If CLXOX bit is 0 , the input clock is CLKOUT1. In all other modes, CLXOX has no effect (don't care x).
8	PRSEN	Prescale clock enable. When set to 1, PRSEN enables the input clock source to the CLKX prescaler CLXCT and extends the scaled CLKX to the ESSP. If reset to 0, the prescaler does not count down as there is no input clock to the counter. The input to CLXCT is CLKOUT1. PRSEN bit functions as a master to all ESSP clocks/modes. All ESSP bits should be preloaded before PRSEN is enabled.
7–5	Transmit FIFO Status	Status of the receive and transmit FIFOs. Define the status of the receive and transmit FIFOs. Each set of 3 bits is capable of indicating five different states that reflect upon the contents of the FIFOs.

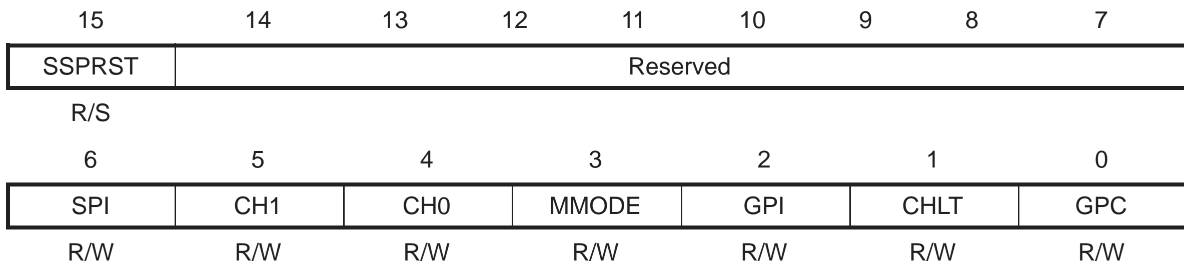
Table 9–7. SSPST Register — I/O address FFF2h Bit Descriptions (Continued)

Bit No.	Name	Function
4–2	Receive FIFO Status	Status of the receive and transmit FIFOs. Define the status of the receive and transmit FIFOs. Each set of 3 bits is capable of indicating five different states that reflect upon the contents of the FIFOs.
1	SGNEX (Sign-Extend)	Sign-extend. When the selected data word size is 8 bits, SGNEX, when set to 1, sign extends the most significant 8 bits of the 16-bit word. If the bit is reset to 0, the most significant 8 bits are filled with zeros.
0	BYTE (8/16 Bit)	Data word size. Defines the data word length as 16 bits or 8 bits. The default value at reset is 0 and selects the 16-bit data word size. The 8-bit data can be received or transmitted by setting bit 0 to 1.

9.10.2 Synchronous Serial Port Multichannel Register (SSPMC)

The SSPMC register is used to select multichannel and 16-bit counter features in the ESSP. Figure 9–12 explains the bit fields used to control the multichannel option on the ESSP.

Figure 9–12. Synchronous Serial Port Multichannel (SSPMC) Register — FFF3h



Note: R = Read, W = Write

Table 9–8. SSPMC Register — FFF3h Bit Descriptions

Bit No.	Name	Function
15	SSPRST	SSPRST resets the current operation of SSP. At reset, SSPRST is 0 and enables normal SSP operation. If set to 1, the SSP resets as follows: a. Resets transmit FIFO pointers and transmit shift register b. Resets receive FIFO pointers and receive shift register c. Prescaler logic reloads the prescaler counters if GPC=0. If GPC=1, there is no reload to prescalers. Resets all logic, except counter logic. d. SSP control register bits (SSPCR) are not affected. However, all status bits are reset.
14–7	Reserved	Reserved
6	SPI	SPI mode bit. SPI, when 1, enables an 8/16-bit pseudo serial peripheral interface (SPI) mode. This mode is available only in burst mode with internal shift clock CLKX. If bit 6 is reset to 0, the SPI mode is disabled. In this mode, CLKX is issued only during the time that data bits are transmitted or received. Data is transmitted/received whenever transmit FIFO has data along with an FSX signal. Prescaled FSX cannot be used in this mode. CLKR and FSR are internally connected to CLKX and FSX, respectively. CLKX pin is normally low in SPI mode. If the CLN bit is enabled in the SSPST register, then the CLKX pin is high between data transmits.
5–4	CH1, CH0	Channel select bit. CH0, CH1 select the number of channels that are available in the multichannel mode. CH0, CH1 have no effect if the MMODE bit is 0. 0 0 Selects one channel with one frame sync pulse FSX1 on FSX pin. The FSX rate is defined only by the FSX prescaler, FSXCT. 0 1 Selects two channels with the second frame sync pulse FSX2 on the CLKR pin (pin 84). Frame sync FSX2 is issued on the second CLKX cycle from the LSB of the first channel. 1 0 Selects three channels with the third frame sync pulse FSX3 on the FSR pin (pin 85). Frame sync FSX3 is issued on the second CLKX cycle from the LSB of the second channel. 1 1 Selects all four channels with the fourth frame sync pulse FSX4 on the IO0 pin (pin 96). Frame sync FSX4 is issued on the second CLKX cycle from the LSB of the third channel. In this mode, the IO0 pin is not available for I/O operation.

Table 9–8. SSPMC Register — FFF3h Bit Descriptions (Continued)

Bit No.	Name	Function
3	MMODE	Multichannel mode bit. MMODE, if reset to the default value 0, deselects the multichannel option on the serial port. If set to 1, MMODE selects the multichannel mode and uses the prescaled frame sync FSX only. In this mode, one or more frame sync pulses are generated on different pins for glueless interface to multiple codecs. The FSX and CLKX signals are internally connected to FSR and CLKR pins respectively. CLKR and FSR pins are available as outputs for generating multichannel frame sync FSX2, FSX3. The fourth channel frame sync (FSX4) is generated on IO0 pin (pin96). In this mode, IO0 is not available as the general purpose I/O pin.
2	GPI	General purpose counter interrupt bit. GPI configures the XINT interrupt of the SSP as the 16-bit counter interrupt. Whenever the 16-bit counter reaches 0, an XINT interrupt is generated instead of a serial port transmit interrupt.
1	CHLT	16-bit counter halt bit. CHLT can be used to stop the 16-bit counter when the prescalers are used as a counter. The default value is 0 and indicates that the counter is counting. A value of 1 stops the counter.
0	GPC	General purpose counter bit. GPC configures the two prescalers CLXCT, FSXCT as a 16-bit counter. When GPC is 1, CLXCT and FSXCT are together used as a 16-bit counter. The input to the counter is either internal CLKOUT1 or CLKX pin clock as defined by CLXOX in SSPST register. In the counter mode the prescalers are not available for ESSP clock scaling. The GPC bit should be 0 if the prescalers are to be used for CLKX and FSX scaling.

9.10.3 Synchronous Serial Port Count Register (SSPCT)

The shift clock CLKX and frame sync FSX can come from external or internal sources. The SSPCR register bits define the source of these signals. The SSPCT register holds two 8-bit prescale counters to provide user-specific shift clock (CLKX) and frame sync clock (FSX). The CLXCT counter is an 8-bit prescaler to divide CLKOUT1. The value of the prescaler output clock is:

CLKOUT1/(2*(CLXCT+1))

CLXCT is the prescale value defined in the SSPCT register bits 7–0. At reset, the CLXCT register value is zero, which makes the CLKX rate equal to (CLKOUT1)/2. This register can be written with any desired 8-bit prescale value. The prescaler functions as a down counter, and the counter value can be read anytime. The input clock source to the CLXCT prescaler can be CLKOUT1 only. PRSEN (bit 8 of the SSPST register) should be set to 1, which enables the input clock to the prescaler.

Once 8-bit prescaler values are written to the register SSPCT, PRSEN must be enabled to start the counter counting down. The prescaler values are

loaded into the counter from the internal buffers only after PRSEN is enabled. Enabling PRSEN should always follow any prescaler update. The prescaler has an internal buffer register that gets updated every time SSPCT is written. After reaching zero, the counter reloads the prescale value from the buffer and counts down. This sequence of reload and count down repeats until PRSEN bit in SSPST is reset to 0. If the PRSEN is reset to 0, the prescaler does not have any input clock source to count down.

FSXCT takes either the CLKX prescaler output or the external CLKX pin clock as its input. This helps to generate a variable frame sync pulse synchronous to CLKX. Most applications require a FSX rate that is a multiple of the CLKX rate. The FSX rate is defined by the equation:

CLKX pin clock/(2*(FSXCT+1))

FSXST bit (bit 12 in SSPST) is set every time FSXCT reaches zero, and can be reset by writing a 1 to the FSXST bit. The 8-bit prescaler FSXCT for FSX also functions in a similar way to the CLKX prescaler CLXCT.

CAUTION

Pay Attention to the FSXCT Value for Serial Channel Configuration

In multichannel mode, the value of FSXCT chosen (for 16-bit data) should be such that there are at least (18 * n) SCLKs between successive frame syncs, where n is the number of serial channels. For example, FSXCT should be greater than or equal to 35 (23h) if four serial channels are configured. For 8-bit data, FSXCT should be greater than or equal to 19 (13h) for four channel configuration. This number is valid for any CLKX and changes only with the number of serial channels configured.

9.10.4 Programmable Internal CLKX and FSX Rates

The device clock CLKOUT1, external shift clock CLKX, and the 8-bit prescalers can provide various CLKX/FSX rates to match several serial interface devices. Interface devices such as CODECs operate in slave mode expecting external shift clock. Table 9–9 provides various shift clock and frame sync rates that can be generated for voice band applications using the prescalers.

Table 9–9. Typical CLKX/FSX Rates and Their Prescaler Values

CLKOUT1	Prescale Value CLXCT Decimal (Hex)	CLKX Rate	Prescale Value FSXCT Decimal (Hex)	FSX Rate	Remarks
40.96 MHz	0	20.48 MHz	255 (FFh)	40 kHz	
	9 (9h)	2.048 MHz	127 (7Fh)	8 kHz	VBAP/combo codec rates
	159 (9Fh)	128 kHz	3 (03h)	16 kHz	
20.48 MHz	0	10.24 MHz	255 (FFh)	20 kHz	
	4 (4h)	2.048 MHz	127 (7Fh)	8 kHz	VBAP/combo codec rates
	159 (9Fh)	64 kHz	3 (03h)	16 kHz	
12.288 x 2 = 24.576 MHz	0	12.288 MHz	383 (17Fh)	16 kHz	
	1h	6.144 MHz	191 (BFh)	16 kHz	
	5h	2.048 MHz	127 (7Fh)	8 kHz	VBAP/combo codec rates
	7h	1.536 MHz	95 (5Fh)	8 kHz	VBAP/combo codec rates
	191 (BFh)	64 kHz	3 (03h)	8 kHz	

9.10.5 Prescalers as General Purpose Counter

The two 8-bit prescalers in the SSPCT register can be used as a single 16-bit down counter. The GPC bit in SSPMC register enables the 16-bit counter mode. When GPC is set to 1, the prescalers are not available for scaling CLKX and FSX. The 16-bit counter can accept either CLKOUT1 clock or CLKX pin clock as its input. The counter value can be read any time and can be stopped by setting CHLT bit in the SSPMC register. The counter flags a status bit FSXST whenever it reaches 0×0000. The counter reloads the counter value after it reaches zero and continues to count down. The FSXST bit is cleared by writing a one to that bit.

Figure 9–13. Synchronous Serial Port Count (SSPCT) Register — FFFBh

Note: R = Read, W = Write

When the prescalers are used as a 16-bit counter, they are not available for prescaling FSX and CLKX. Two options are possible in the 16-bit counter mode (GPC = 1).

- Option 1: Internal CLKX (MCM = 1)

  When CLXOX = 1, input to counter is CLKX which is CLKOUT1/2, since the prescalers are not operating.

  When CLXOX = 0, input to the counter is CLKOUT1.

- Option 2: External CLKX (MCM =0)

  When CLXOX = 1, input to counter is the CLKX pin.

  When CLXOX = 0, input to counter is CLKOUT1.

9.11 ESSP Register Programming Considerations

All standard SSP features can be configured by programming the ESSP register (SSPCR) alone. This provides compatibility to the existing codes for standard SSP in TMS320C203. However, if ESSP features such as multichannel mode, prescaled frame sync, and shift clocks are desired, it is necessary to initialize ESSP registers (SSPCT, SSPMC, and SSPST). It is recommended that registers SSPCT and SSPMC are initialized first, followed by the SSPST register. The prescalers are enabled only after the PRSEN bit (bit 8 in SSPST) is set to 1. It is essential that the other registers be preloaded before enabling the PRSEN bit in the SSPST register.

9.11.1 ESSP Register Initialization

While changing CLKN or FSN bits, initialize the SSPST register in two steps:

1) Load the SSPST registers bits with PRSEN bit 0.

2) Provide at least one CLKX cycle delay before setting PRSEN bit.

This helps internal synchronization of all the clocks (FSX/CLKX). This also makes the prescalers and the clock circuit respond to the stable clock (FSX/CLKX, FSR,CLKR) edges. However, in any initialization sequence, the prescaler clocks are stable after the first reload of the prescaler counters.

9.11.2 Prescaler Values in Multichannel Mode

Considerable attention must be paid in choosing the value of FSXCT in multichannel mode. For 16-bit data, successive frame sync pulses occur 18 SCLKs after the previous frame sync pulse. In the multichannel mode, if all 4 channels are used, a new data word is transmitted after a period of 72 SCLKs for a given channel. (A new frame sync can occur only after 72 SCLKs.) This is the minimum requirement. The minimum value for FSXCT can be easily found from the formula for calculating the FSX rate. This is done by applying the condition that two frame syncs for a given channel must be separated by at least (N * 18) SCLKs, where N is the number of channels in the multi-channel mode. This condition is applicable for 16-bit mode, where successive frame syncs are separated by 18 SCLKs. In 8-bit mode, the frame syncs are separated by 10 SCLKs. PRSEN must be 1 for the FSXCT prescaler to operate correctly.

Figure 9–14. Typical Four-Channel Codec Interface

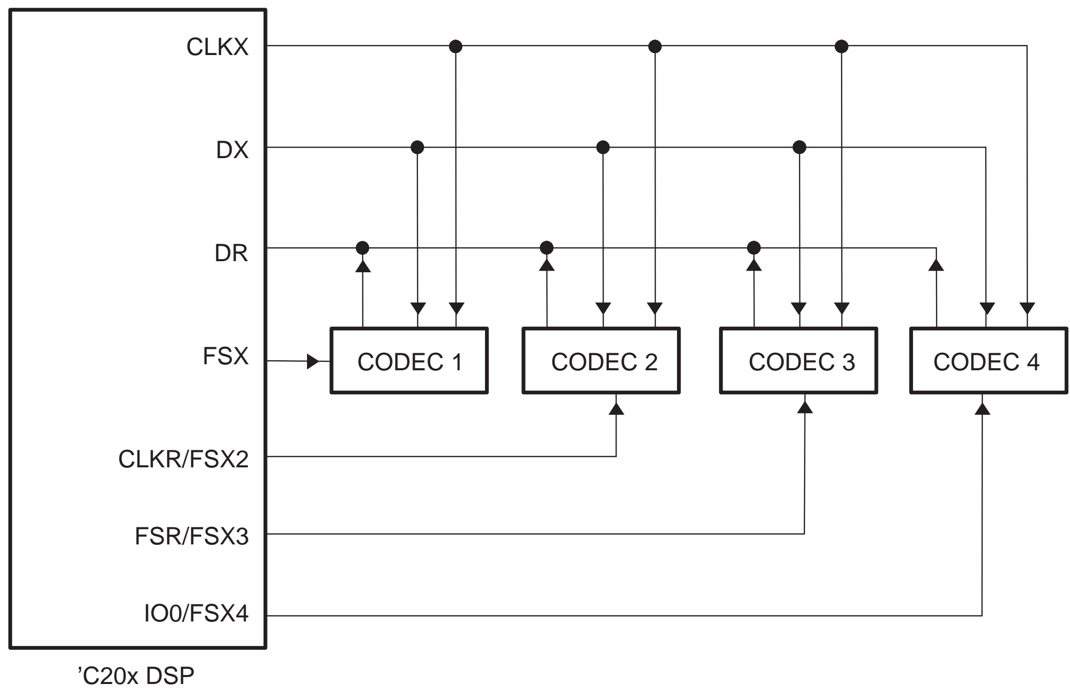
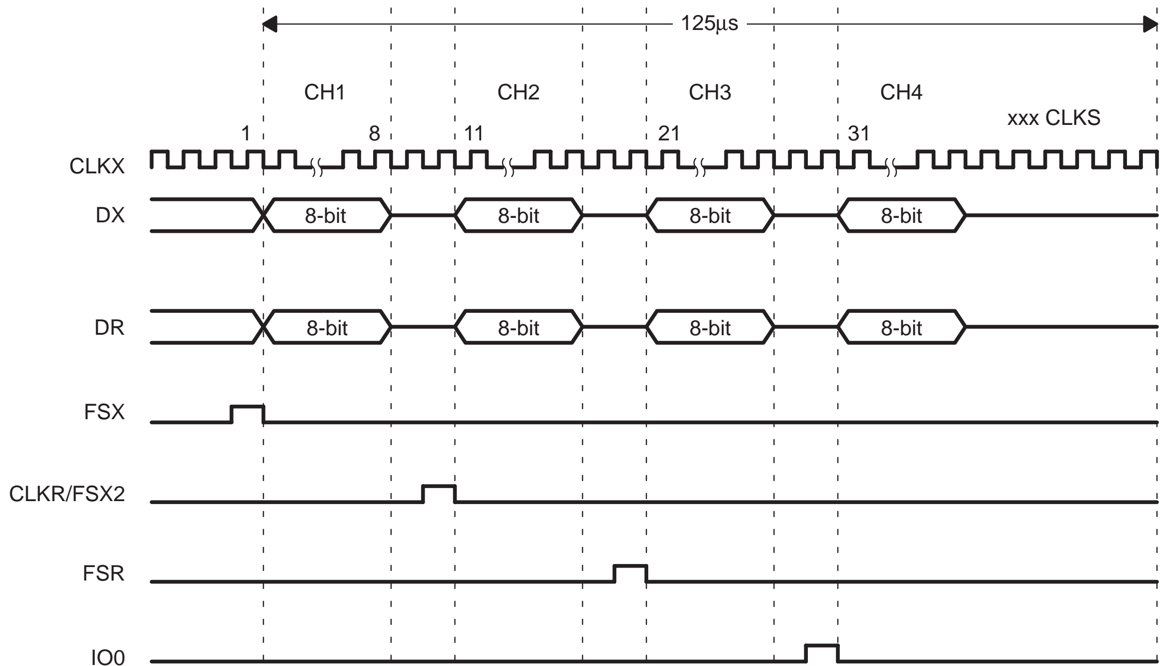
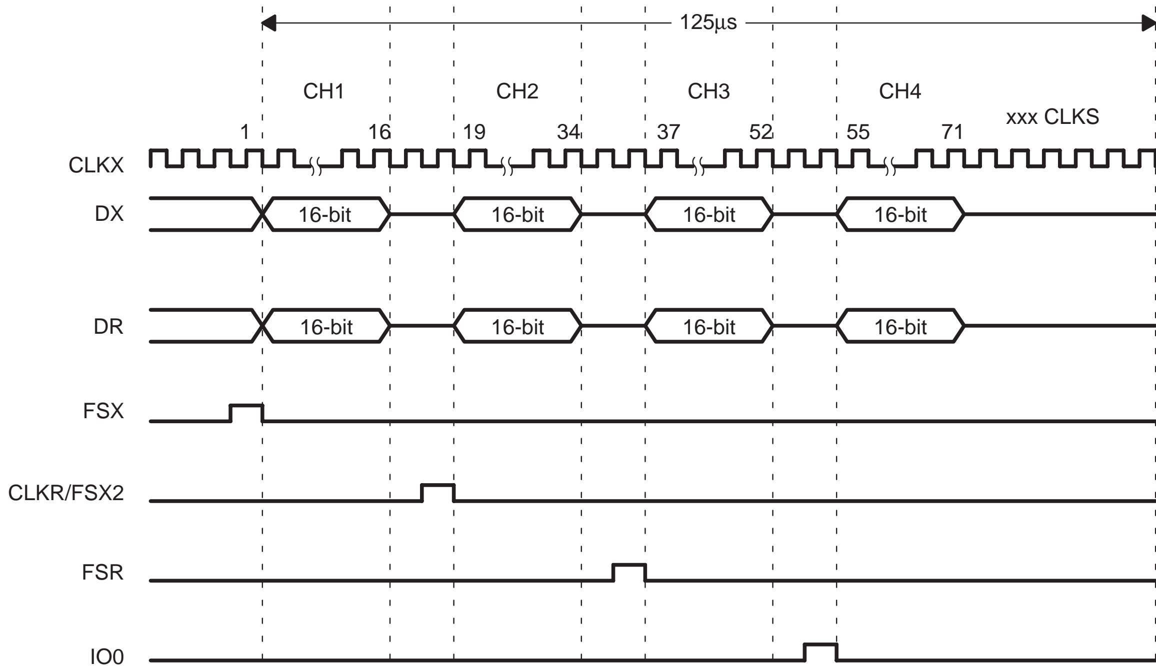


Figure 9–15. Four-Channel 8-Bit CODEC† Interface Timing Example



† CODEC – coder-decoder devices such as COMBO/VBAP type of telephony codecs

Figure 9–16. Four-Channel 16-Bit CODEC† Interface Timing Example



† CODEC – coder-decoder devices such as COMBO/VBAP type of telephony codecs

9.11.3 ESSP Serial Port Configurations

The ESSP port can be configured for two modes of operation, burst and continuous, by setting bits in the SSPCR, SSPMC, and SSPST registers. Table 9–10 lists the SSP and ESSP functions, by option number, available for both modes. Note that in continuous mode, the Multichannel and SPI functions (options 10, 11, 12) are not available. Table 9–11 shows burst mode, and Table 9–12 shows continuous mode.

Table 9–10. Options/Functions for Burst Mode and Continuous Mode

ESSP Configuration		Register Bits						
		CLKX			FSX			
Option	**Function**	**E**	**I**	**P**	**E**	**I**	**P**	**FIFO-rate[†]**
1	SSP RESET	–	–	–	–	–	–	–
2	SSP option[‡]	✓			✓			
3	SSP option with FSXCT[‡]	✓				✓	✓	
4	SSP option[‡]	✓				✓		✓
5	SSP option[‡]		✓		✓			
6	SSP option with CLXCT[‡]		✓	✓	✓			
7	SSP option with 8-bit prescalers[‡]		✓	✓		✓	✓	
8	SSP option[‡]		✓	✓		✓		✓
9	SSP option with CLKXCT[‡]		✓	✓		✓		✓
10	Multichannel[§]		✓	✓		✓	✓	
11	Multichannel[§]	✓				✓	✓	
12	SPI[§]		✓	✓		✓		✓
13	Counter and SSP		✓			✓		✓
14	Counter and SSP		✓		✓			
15	Counter and SSP	✓				✓		✓
16	Counter and SSP	✓			✓			

Legend: **E** - External **I** - Internal **P** - Prescaled

† TXFIFO WRITE RATE: In this state, the frame sync is issued along with each word transmitted from the TXFIFO.
‡ SSP Option refers to all features of the standard SSP – without the use of the ESSP register bits. These options differ based on CLKX and FSX source.
§ Multichannel and SPI functions (options 10, 11, 12) are not available in continuous mode.

Table 9–11. Serial Port Configuration – Burst Mode

Options	SSPCR Register			SSPMC Register								SSPST Register												
	FSM	MCM	TXM	SSPRST	SPI	CHB1	CHB0	MMODE	GPI	CHLT	GPC	FSN	FSXOX	FSXST	CLN	CLXOX	PRSEN	BYTE	CLKX	FSX	CLXCT	CLKX rate	FSXCT	FSX rate
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E	E	Not used	–	Not used	–
2	1	0	0	0	0	X	X	0	0	X	0	0	X	0	0	X	0	0/1	E	E	Not used	E CLKX	Not used	E FSX
3	1	0	1	0	0	X	X	0	0	X	0	0	1	0	0	X	1	0/1	E	I	Not used	E CLKX	Used P FSX	I FSX by FSXCT
4	1	0	1	0	0	X	X	0	0	X	0	0	0	0	0	X	X	0/1	E	I	Not used	E CLKX	Not used	I FSX
5	1	1	0	0	0	X	X	0	0	X	0	0	X	0	0	X	0	0/1	I	E	Not used	1/2 C1	Not used	I FSX by TX FIFO write
6	1	1	0	0	0	X	X	0	0	X	0	0	X	0	0	X	1	0/1	I	E	Used CLKX	1/2 C1 or P	Not used	E FSX
7	1	1	1	0	0	X	X	0	0	X	0	0	1	0	0	X	1	0/1	I	I	Used CLKX	1/2 C1 or P	Used	E FSX
8	1	1	1	0	0	X	X	0	0	X	0	0	0	0	0	X	0	0/1	I	I	Not used	1/2 C1	Not used	Def by FSX/P FSXCT
9	1	1	1	0	0	X	X	0	0	X	0	0	0	0	0	X	1	0/1	I	I	Used CLKX	1/2 C1 or P	Not used	Def by write to TX FIFO
10	1	1	1	0	0	0/1	0/1	1	0	0	0	0	1	0	0	X	1	0/1	I	I	Used	1/2 C1 or P	Used	Def by write to TX FIFO
11	1	0	1	0	0	0/1	0/1	1	0	0	0	0	1	0	0	X	1	0/1	E	I	Not used	E CLKX	Used	FSX1 def by FSXCT†
12	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	X	1	0/1	I	I	Used	1/2 C1 or P	Not used	FSX1 def by FSXCT†
13	1	1	1	0	0	0	0	0	u	u	1	0	0	0	0	0/1	1	0/1	I	I	U16	1/2 C1	U16	Def by write to TX FIFO
14	1	1	0	0	0	0	0	0	u	u	1	0	X	0	0	0/1	1	0/1	I	E	U16	1/2 C1	U16	E
15	1	0	1	0	0	0	0	0	u	u	1	0	0	0	0	0/1	1	0/1	E	I	U16	E	U16	Def by write to TX FIFO
16	1	0	0	0	0	0	0	0	u	u	1	0	X	0	0	0/1	1	0/1	E	E	U16	E	U16	E

Legend: **E** - External, **I** - Internal, **1/2 C1** - 1/2 CLKOUT1, **P** - Prescaled, **U16** - Used by 16-bit Counter, **Def** - Defined, **u** - Defines other functions in the selected mode. 0 and 1 are valid options. **X** - DON'T CARE, does not affect selected mode. Replace X with 0 while writing to registers.

†FSXCT defines FSX rate to be greater than (18x4) SCLKs for 16-bit data and (10x4) SCLKs for 8-bit data, or the FSX rate will be incorrect.

Table 9–12. Serial Port Configuration – Continuous Mode

Options	SSPCR Register FSM	MCM	TXM	SSPMC Register SSPRST	SPI	CHB1	CHB0	MMODE	GPI	CHLT	GPC	SSPST Register FSN	FSXOX	FSXST	CLN	CLXOX	PRSEN	BYTE	CLKX	FSX	CLXCT	CLKX rate	FSXCT	FSX rate
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E	E	Not used	–	Not used	–
2	0	0	0	0	0	X	X	0	0	X	0	0	X	0	0	0	0	0/1	E	E	Not used	E CLKX only	Not used	E
3	0	0	1	0	0	X	X	0	0	X	0	0	1	0	0	X	1	0/1	E	I	Not used	E CLKX only	Used I FSX	I FSX def by FSXCT
4	0	0	1	0	0	X	X	0	0	X	0	0	0	0	0	X	X	0/1	E	I	Not used	E CLKX	Not used	TX FIFO write rate
5	0	1	0	0	0	X	X	0	0	X	0	0	X	0	0	X	0	0/1	I	E	Not used	1/2 C1	Not used	E FSX
6	0	1	0	0	0	X	X	0	0	X	0	0	X	0	0	X	1	0/1	I	E	Used CLKX	1/2 C1 or P	Not used	E FSX
7	0	1	1	0	0	X	X	0	0	X	0	0	1	0	0	X	1	0/1	I	I	Used CLKX	1/2 C1 or P	Used	Def by FSX P
8	0	1	1	0	0	X	X	0	0	X	0	0	0	0	0	X	0	0/1	I	I	Not used	1/2 C1	Not used	Def by write to TX FIFO
9	0	1	1	0	0	X	X	0	0	X	0	0	0	0	0	X	1	0/1	I	I	Used CLKX	1/2 C1 or P	Not used	Def by write to TX FIFO
10	For options 10, 11, and 12,																							
11	there is no Multichannel or SPI function in the Continuous Mode of the SSP.																							
12	(FSM bit is a Don’t Care for this mode.)																							
13	0	1	1	0	0	0	0	0	u	u	1	0	0	0	0	0/1	1	0/1	I	I	U16	1/2 C1	U16	Def by write to TX FIFO
14	0	1	0	0	0	0	0	0	u	u	1	0	X	0	0	0/1	1	0/1	I	E	U16	1/2 C1	U16	E
15	0	0	1	0	0	0	0	0	u	u	1	0	0	0	0	0/1	1	0/1	E	I	U16	E	U16	Def by write to TX FIFO
16	0	0	0	0	0	0	0	0	u	u	1	0	X	0	0	0/1	1	0/1	E	E	U16	E	U16	E

Legend: **E** - External, **I** - Internal, **1/2 C1** - 1/2 CLKOUT1, **P** - Prescaled, **U16** - Used by 16-bit Counter, **Def** - Defined, **u** - Defines other functions in the selected mode. 0 and 1 are valid options. **X** - DON’T CARE, does not affect selected mode. Replace X with 0 while writing to registers.

†FSXCT defines FSX rate to be greater than (18x4) SCLKs for 16-bit data and (10x4) SCLKs for 8-bit data, or the FSX rate will be incorrect.

Chapter 10

Asynchronous Serial Port

The 'C20x has an asynchronous serial port that can be used to transfer data to and from other devices. The port has several important features:

- Full-duplex transmit and receive operations at the maximum transfer rate
- Data-word length of eight bits for both transmit and receive
- Capability for using one or two stop bits
- Double buffering in all modes to transmit and receive data
- Adjustable baud rate of up to 250,000 10-bit characters per second
- Automatic baud-rate detection logic

For examples of program code for the asynchronous serial port, see Appendix D, *Program Examples*.

Topic **Page**

10.1 Overview of the Asynchronous Serial Port 10-2

10.2 Components and Basic Operation 10-3

10.3 Controlling and Resetting the Port 10-7

10.4 Transmitter Operation 10-19

10.5 Receiver Operation 10-20

10.1 Overview of the Asynchronous Serial Port

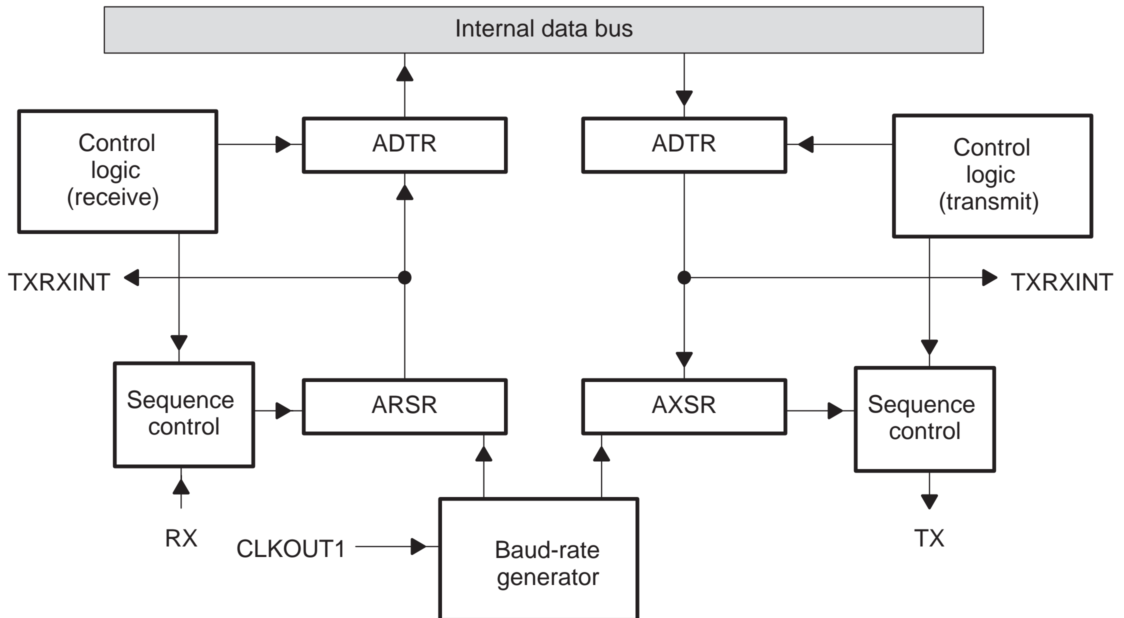
The on-chip asynchronous serial port (ASP) provides easy serial data communication between host CPUs and the ’C20x or between two ’C20x devices. The asynchronous mode of data communication is often referred to as UART (universal asynchronous receive and transmit). For transmissions, data written to a transmit register is converted from an 8-bit parallel form to a 10- or 11-bit serial form (the eight bits preceded by one start bit and followed by one or two stop bits). Each of the ten or eleven bits is transmitted sequentially (LSB first) to a transmit pin. For receptions, data is received one bit at a time (LSB first) at a receive pin (one start bit, eight data bits, and one or two stop bits). The received bits are converted from serial form to parallel form and stored in the lower eight bits of a 16-bit receive register. Errors in data transfers are indicated by flags and/or interrupts. The asynchronous serial port is reset 16 CLKOUT1 cycles after the rising edge of the reset pin, during device reset.

The maximum rate for transmissions and receptions is determined by the rate of the internal baud clock, which operates at a fraction of the rate of CLKOUT1. The exact fraction is determined by the value in the 16-bit programmable baud-rate divisor register (BRD). For receptions, you may enable (through software) the auto-baud detection logic, which allows the ASP to lock to the incoming data rate.

10.2 Components and Basic Operation

Figure 10–1 shows the main components of the asynchronous serial port.

Figure 10–1. Asynchronous Serial Port Block Diagram



10.2.1 Signals

Two types of signals are used in asynchronous serial port (ASP) operations:

- ❑ Data signal. A data signal carries data from the transmitter to the receiver. Data is sent through the transmit pin (TX) on the transmitter and accepted through the receive pin (RX) on the receiver. One-way serial port transmission requires one data signal; two-way transmission requires two data signals.

- ❑ Handshake signal.The data transfer can be improved by using bits IO0–IO3 of the ASP control register (ASPCR) for handshaking.

Data is transmitted on a character-by-character basis. Each data frame contains a start bit, eight data bits, and one or two stop bits. The transmit and receive sections are both double-buffered to allow continuous data transfers.

The pins used by the asynchronous serial port are summarized in Table 10–1. Each of these pins has an associated signal with the same name.

Table 10–1. Asynchronous Serial Port Interface Pins

Pin Name	Description
TX	*Asynchronous serial port data transmit pin.* Transmits serial data from the asynchronous serial port transmit shift register (AXSR).
RX	*Asynchronous serial port data receive pin.* Receives serial data into the asynchronous serial port receive shift register (ARSR).
IO0	*General purpose I/O pin 0.* Can be used for general purpose I/O or for handshaking by the UART.
IO1	*General purpose I/O pin 1.* Can be used for general purpose I/O or for handshaking by the UART.
IO2	*General purpose I/O pin 2.* Can be used for general purpose I/O or for handshaking by the UART.
IO3	*General purpose I/O pin 3.* Can be used for general purpose I/O or for handshaking by the UART.

10.2.2 Baud-Rate Generator

The baud-rate generator is a clock generator for the asynchronous serial port. The output rate of the generator is a fraction of the CLKOUT1 rate and is controlled by a 16-bit register, BRD, that you can read from and write to at I/O address FFF7h. For a CLKOUT1 frequency of 40 MHz, the baud-rate generator can generate baud rates as high as 2.5 megabits/s (250,000 characters/s) and as low as 38.14 bits/s (3.81 characters/s).

10.2.3 Registers

Four on-chip registers allow you to transmit and receive data and to control the operation of the port:

- Asynchronous data transmit and receive register (ADTR). The ADTR is a 16-bit read/write register for transmitting and receiving data. Data written to the lower eight bits of the ADTR is transmitted by the asynchronous serial port. Data received by the port is read from the lower eight bits of the ADTR. The upper byte is read as zeros. The ADTR is an on-chip register located at address FFF4h in I/O space.
- Asynchronous serial port control register (ASPCR). The ASPCR, at I/O address FFF5h, contains bits for setting port modes, enabling or disabling the automatic baud-rate detection logic, selecting the number of stop bits, enabling or disabling interrupts, setting the default level on the TX pin, configuring pins IO3–IO0, and resetting the port. Section 10.3.1 gives a detailed description of the ASPCR.

- I/O status register (IOSR). Bits in the IOSR indicate detection of the incoming baud rate, various error conditions, the status of data transfers, detection of a break on the RX pin, the status of pins IO3–IO0, and detection of changes on pins IO3–IO0. The IOSR is at address FFF6h in I/O space. For detailed descriptions of the bits in the IOSR, see section 10.3.2.
- Baud-rate divisor register (BRD). The 16-bit value in the BRD is a divisor used to determine the baud rate for data transfers. BRD (at address FFF7h in I/O space) is either loaded by software or is loaded by the port when the automatic baud-rate detection logic is enabled and samples the incoming baud rate. Section 10.3.3 describes how to determine the BRD value that will produce the desired baud rate.

Two other registers (not accessible to a programmer) control transfers between the ADTR and the pins:

- Asynchronous serial port transmit shift register (AXSR). During transmissions, each data character is transferred from the ADTR to the AXSR. The AXSR then shifts the character out (LSB first) through the TX pin.
- Asynchronous serial port receive shift register (ARSR). During receptions, each data character is accepted, one bit at a time (LSB first), at the RX pin and shifted into the ARSR. The ARSR then transfers the character to the ADTR.

10.2.4 Interrupts

The asynchronous serial port has one hardware interrupt (TXRXINT), which can be generated by various events (described in section 10.3.6). TXRXINT leads the CPU to interrupt vector location 000Ch in program memory. The branch at that location should lead to an interrupt service routine that identifies the cause of the interrupt and then acts accordingly. TXRXINT has a priority level of 9 (1 being highest).

TXRXINT is a maskable interrupt controlled by the interrupt mask register (IMR) and interrupt flag register (IFR).

Note:

To avoid a double interrupt from the ASP, clear the IFR bit (TXRXINT) in the corresponding interrupt service routine, just before returning from the routine.

10.2.5 Basic Operation

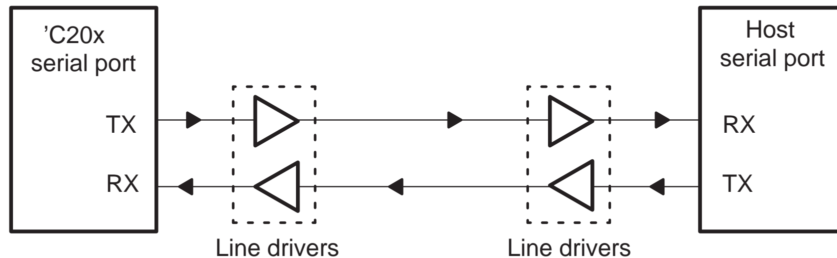
Figure 10–2 shows a typical serial link between a ’C20x device and any host CPU. In this mode of communication, any 8-bit character can be transmitted

or received serially by way of the transmit data pin (TX) or the receive data pin (RX), respectively. The data transmitted or received through the TX and RX pins will be at TTL level. However, if the hosts are separated by a few feet or more, the serial data lines must be buffered through line-drivers (RS-232 or RS-485, depending on the application).

When an 8-bit character is written into the lower eight bits of the ADTR, the data, in parallel form, is converted into a 10- or 11-bit character with one start bit and one or two stop bits. This new 10- or 11-bit character is then converted into a serial data stream and transmitted through the TX pin one bit at a time. The bit duration is determined by the baud clock rate. The baud-rate divisor register (BRD) is programmable and takes a 16-bit value, providing all the industry-standard baud rate values.

Similarly, if a 10- or 11-bit data stream reaches the RX pin, the serial port samples the bit at the transmitted baud rate and converts the serial stream into an 8-bit parallel data character. The received 8-bit character is stored in the lower eight bits of the ADTR.

Figure 10–2. Typical Serial Link Between a 'C20x Device and a Host CPU



10.3 Controlling and Resetting the Port

The asynchronous serial port is programmed through three on-chip registers mapped to I/O space: the asynchronous serial port control register (ASPCR), the I/O status register (IOSR), and the baud-rate divisor register (BRD). This section describes the contents of each of these registers and also explains the use of associated control features.

10.3.1 Asynchronous Serial Port Control Register (ASPCR)

The ASPCR controls the operation of the asynchronous serial port. Figure 10–3 shows the fields in the 16-bit memory-mapped ASPCR and bit descriptions follow the figure. All of the bits in the register are read/write, with the exception of the reserved bits (12–10). The ASPCR is an on-chip register mapped to address FFF5h in I/O space.

Figure 10–3. Asynchronous Serial Port Control Register (ASPCR) — I/O-Space Address FFF5h

15	14	13	12	11	10	9	8
FREE	SOFT	URST	Reserved			DIM	TIM
R/W–0	R/W–0	R/W–0	0			R/W–0	R/W–0

7	6	5	4	3	2	1	0
RIM	STB	CAD	SETBRK	CIO3	CIO2	CIO1	CIO0
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Note: 0 = Always as zeros; R=Read access; W=Write access; value following dash (–) is value after reset.

Table 10–2. ASPCR — I/O Space Address FFF5h Bit Descriptions

Bit No.	Name	Function
15	FREE	This bit sets the port to function in emulation or run mode.
		0 Emulation mode is selected. SOFT then determines which emulation mode is enabled.
		1 Free run mode is selected.
14	SOFT	This bit is enabled when the FREE bit is 0. It determines the emulation mode.
		0 Process stops immediately.
		1 Process stops after word completion.

Table 10–2. ASPCR — I/O Space Address FFF5h Bit Descriptions (Continued)

Bit No.	Name	Function
13	URST	Reset asynchronous serial port bit. URST is used to reset the asynchronous serial port. At reset, URST = 0. 0 The port is in reset. 1 The port is enabled.
12–10	Reserved	Always read as 0s.
9	DIM	Delta interrupt mask. DIM selects whether or not delta interrupts are asserted on the TXRXINT interrupt line. A delta interrupt is generated by a change on one of the general-purpose I/O pins (IO3, IO2, IO1, or IO0). 0 Disables delta interrupts. 1 Enables delta interrupts.
8	TIM	Transmit interrupt mask. TIM selects whether transmit interrupts are asserted on the TXRXINT interrupt line. A transmit interrupt is generated by THRE (transmit register empty indicator in the IOSR) when the transmit register (ADTR) empties. 0 Disables transmit interrupts. 1 Enables transmit interrupts.
7	RIM	Receive interrupt mask. RIM selects whether receive interrupts are asserted on the TXRXINT interrupt line. A receive interrupt is generated by one of these indicators in the IOSR: BI (break interrupt), FE (framing error), OE (overflow error), or DR (data ready). 0 Disables receive interrupts. 1 Enables receiver interrupts.
6	STB	Stop bit selector. STB selects the number of stop bits used in transmission and reception. 0 One stop bit is used in transmission and reception. This is the default value at reset. 1 Two stop bits are used in transmission and reception.
5	CAD	Calibrate A detect bit. CAD is used to enable and disable automatic baud-rate alignment (auto-baud alignment). 0 Disables auto-baud alignment. 1 Enables auto-baud alignment.

Table 10–2. ASPCR — I/O Space Address FFF5h Bit Descriptions (Continued)

Bit No.	Name	Function
4	SETBRK	Set break bit. Selects the output level of TX when the port is not transmitting. 0 The TX output is forced high when the port is not transmitting. 1 The TX output is forced low when the port is not transmitting.
3	CIO3	Configuration bit for IO3. CIO3 configures I/O pin 3 (IO3) as an input or as an output. 0 IO3 is configured as an input. This is the default value at reset. 1 IO3 is configured as an output.
2	CIO2	Configuration bit for IO2. CIO2 configures I/O pin 2 (IO2) as an input or as an output. 0 IO2 is configured as an input. This is the default value at reset. 1 IO2 is configured as an output.
1	CIO1	Configuration bit for IO1. CIO1 configures I/O pin 1 (IO1) as an input or as an output. 0 IO1 is configured as an input. This is the default value at reset. 1 IO1 is configured as an output.
0	CIO0	Configuration bit for IO0. CIO0 configures I/O pin 0 (IO0) as an input or as an output. 0 IO0 is configured as an input. This is the default value at reset. 1 IO0 is configured as an output.

10.3.2 I/O Status Register (IOSR)

The IOSR returns the status of the asynchronous serial port and of I/O pins IO0–IO3. The IOSR is a 16-bit, on-chip register mapped to address FFF6h in I/O space. Figure 10–4 shows the fields in the IOSR, and bit descriptions follow the figure.

Figure 10–4. I/O Status Register (IOSR) — I/O-Space Address FFF6h

15	14	13	12	11	10	9	8
Reserved	ADC	BI	TEMT	THRE	FE	OE	DR
0	R/W1C–0	R/W1C–0	R–1	R–1	R/W1C–0	R/W1C–0	R–0

7	6	5	4	3	2	1	0
DIO3	DIO2	DIO1	DIO0	IO3	IO2	IO1	IO0
R/W1C–x	R/W1C–x	R/W1C–x	R/W1C–x	R/W†–x	R/W†–x	R/W†–x	R/W†–x

Note: 0 = Always read as 0; R=Read access; W1C=Write 1 to this bit to clear it to 0; W = Write access; value following dash (–) is value after reset (x means value not affected by reset).

† This bit can be written to only when it is configured as an output by the corresponding CIO bit in the ASPCR.

Table 10–3. IOSR — I/O Space Address FFF6h Bit Descriptions

Bit No.	Name	Function
15	Reserved	Always read as 0.
14	ADC	A detect complete bit. If the CAD bit of the ASPCR is 1 and the character *A* or *a* is received in the ADTR, ADC is set to 1. The character *A* or *a* remains in the ADTR after it has been detected. To avoid an overrun error when the next character arrives, the ADTR should be read immediately after ADC is set. 0 *A* or *a* has not been detected. No receive interrupt (TXRXINT) will be generated. 1 *A* or *a* has been detected. If the CAD bit of the ASPCR is also 1, a receive interrupt (TXRXINT) will be generated, regardless of the values of the DIM, TIM, and RIM bits of the ASPCR. For as long as ADC = 1 and CAD = 1, a receive interrupt will occur.
13	BI	Break interrupt indicator. BI = 1 indicates that a break has been detected on the RX pin. Write a 1 to this bit to clear it to 0. BI is also cleared to 0 at reset. A break on the RX pin also generates an interrupt (TXRXINT).

Table 10–3. IOSR — I/O Space Address FFF6h Bit Descriptions (Continued)

Bit No.	Name	Function
12	TEMT	Transmit empty indicator. TEMT = 1 indicates whether the transmit register (ADTR) and/or transmit shift register (AXSR) are full or empty. This bit is set to 1 on reset. 0 The ADTR and/or AXSR are full. 1 The ADTR and the AXSR are empty; the ADTR is ready for a new character to transmit.
11	THRE	Transmit register (ADTR) empty indicator. THRE is set to 1 when the contents of the transmit register (ADTR) are transferred to the transmit shift register (AXSR). THRE is reset to 0 by the loading of the transmit register with a new character. A device reset sets THRE to 1. The emptying of the ADTR also generates an interrupt (TXRXINT). 0 The transmit register is not empty. Port operation is normal. 1 The transmit register is empty, indicating that it is ready to be loaded with a new character.
10	FE	Framing error indicator. FE indicates whether a valid stop bit has been detected during reception. Clear the FE bit to 0 by writing a 1 to it. It is also cleared to 0 on reset. A framing error also generates an interrupt (TXRXINT). 0 No framing error is detected. Port operation is normal. 1 The character received did not have a valid (logic 1) stop bit.
9	OE	Receive register (ADTR) overrun indicator. OE indicates whether an unread character has been overwritten. Clear the OE bit to 0 by writing a 1 to it. It is also cleared to 0 on reset. The occurrence of overrun also generates an interrupt (TXRXINT). 0 No overrun error is detected. The port is operating normally. 1 The last character in the ADTR was not read before the next character overwrote it.

Table 10–3. IOSR — I/O Space Address FFF6h Bit Descriptions (Continued)

Bit No.	Name	Function
8	DR	Data ready indicator for the receiver. This bit indicates whether a new character has been received in the ADTR. This bit is automatically cleared to zero when the receive register (ADTR) is read or when the device is reset. The reception of a new character into the ADTR also generates an interrupt (TXRXINT). 0 The receive register (ADTR) is empty. 1 A character has been completely received and should be read from the receive register (ADTR).
7	DIO3	Change detect bit for IO3. DIO3 indicates whether a change has occurred on the IO3 pin. A change can be detected only when IO3 is configured as an input by the CIO3 bit of the ASPCR (CIO3 = 0) and the serial port is enabled by the URST bit of the ASPCR (URST = 1). Writing a 1 to DIO3 clears it to 0. The detection of a change on the IO3 pin also generates an interrupt (TXRXINT). 0 No change is detected on IO3. 1 A change is detected on IO3.
6	DIO2	Change detect bit for IO2. DIO2 indicates whether a change has occurred on the IO2 pin. A change can be detected only when IO2 is configured as an input by the CIO2 bit of the ASPCR (CIO2 = 0) and the serial port is enabled by the URST bit of the ASPCR (URST = 1). Writing a 1 to DIO2 clears it to 0. The detection of a change on the IO2 pin also generates an interrupt (TXRXINT). 0 No change is detected on IO2. 1 A change is detected on IO2.
5	DIO1	Change detect bit for IO1. DIO1 indicates whether a change has occurred on the IO1 pin. A change can be detected only when IO1 is configured as an input by the CIO1 bit of the ASPCR (CIO1 = 0) and the serial port is enabled by the URST bit of the ASPCR (URST = 1). Writing a 1 to DIO1 clears it to 0. The detection of a change on the IO1 pin also generates an interrupt (TXRXINT). 0 No change is detected on IO1. 1 A change is detected on IO1.

Table 10–3. IOSR — I/O Space Address FFF6h Bit Descriptions (Continued)

Bit No.	Name	Function
4	DIO0	Change detect bit for IO0. DIO0 indicates whether a change has occurred on the IO0 pin. A change can be detected only when IO0 is configured as an input by the CIO0 bit of the ASPCR (CIO0 = 0) and the serial port is enabled by the URST bit of the ASPCR (URST = 1). Writing a 1 to DIO0 clears it to 0. The detection of a change on the IO0 pin also generates an interrupt (TXRXINT). 0 No change is detected on IO0. 1 A change is detected on IO0.
3	IO3	Status bit for IO3. When the IO3 pin is configured as an input (by the CIO3 bit of the ASPCR), this bit reflects the current level on the IO3 pin. 0 The IO3 signal is low. 1 The IO3 signal is high.
2	IO2	Status bit for IO2. When the IO2 pin is configured as an input (by the CIO2 bit of the ASPCR), this bit reflects the current level on the IO2 pin. 0 The IO2 signal is low. 1 The IO2 signal is high.
1	IO1	Status bit for IO1. When the IO1 pin is configured as an input (by the CIO1 bit of the ASPCR), this bit reflects the current level on the IO1 pin. 0 The IO1 signal is low. 1 The IO1 signal is high.
0	IO0	Status bit for IO0. When the IO0 pin is configured as an input (by the CIO0 bit of the ASPCR), this bit reflects the current level on the IO0 pin. 0 The IO0 signal is low. 1 The IO0 signal is high.

Note: If IO0–3 pins have been configured as outputs, IO0–3 bits can be written with either a 1 or 0 to reflect on the I/O pins (0–3) respectively.

10.3.3 Baud-Rate Divisor Register (BRD)

The baud rate of the asynchronous serial port can be set to many different rates by means of the BRD, an on-chip register located at address FFF7h in I/O space. Equation 10–1 shows how to set the BRD value to get the desired baud rate. When the BRD contains 0, the ASP will not transmit or receive any character. At reset, BRD = 0001h.

Equation 10–1. Value Needed in the BRD

$$\text{BRD value in decimal} = \frac{\text{CLKOUT1 frequency}}{16 \times \text{desired baud rate}}$$

Table 10–4 lists common baud rates and the corresponding hexadecimal value that should be in the BRD for a given CLKOUT1 frequency.

Table 10–4. Common Baud Rates and the Corresponding BRD Values

	BRD Value in Hexadecimal		
Baud Rate	**CLKOUT1 = 20 MHz (50 ns)**	**CLKOUT1 = 28.57 MHz (35 ns)**	**CLKOUT1 = 40 MHz (25 ns)**
1200	0411	05CC	0823
2400	0208	02E6	0411
4800	0104	0173	0208
9600	0082	00B9	0104
19200	0041	005C	0082

10.3.4 Using Automatic Baud-Rate Detection

The ASP contains auto-baud detection logic, which allows the ASP to lock to the incoming data rate. The following steps explain the sequence by which the detection logic could be implemented:

1) Enable auto-baud detection by setting the CAD bit in the ASPCR to 1 and ADC bit in the IOSR to zero.

2) Receive from a host the ASCII character *A* or *a* as the first character, at any desired baud rate definable in the BRD register. If the first character received is *A* or *a*, the serial port will lock to the incoming baud rate (the rate of the host), and the BRD register will be updated to the incoming baud rate value.

3) Baud-rate detection is indicated by a TXRXINT interrupt (mapped to vector location 000Ch) if TXRXINT is unmasked in the interrupt mask register and is globally enabled by the INTM bit of status register ST0. This interrupt occurs regardless of the values of the DIM, TIM, and RIM bits in the ASPCR.

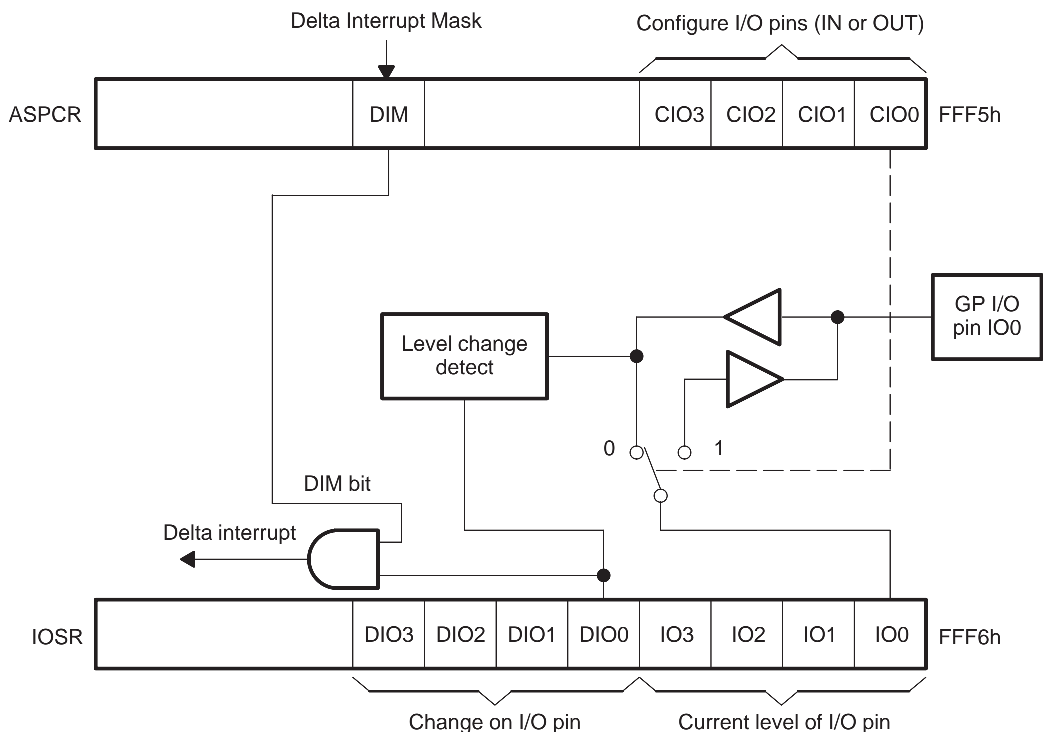
4) Following the baud detection interrupt, the ADTR should be read to clear the *A* or *a* character from the receive buffer. If the ADTR is not cleared, any subsequent character received will set the OE bit in the IOSR, indicating an overrun error.

5) Once the baud rate is detected, both the CAD and ADC bits must be cleared; write 0 to CAD and write 1 to ADC. If CAD is not cleared, the auto baud-detection logic will try to lock to the incoming character speed. In addition, for as long as ADC = 1 and CAD = 1, receive interrupts will be generated.

10.3.5 Using I/O Pins IO3, IO2, IO1, and IO0

Pins IO3, IO2, IO1, and IO0 can be individually configured as inputs or outputs and can be used as handshake control for the asynchronous serial port or as general-purpose I/O pins. They are software-controlled through the asynchronous serial port control register (ASPCR) and the I/O status register (IOSR), as shown in Figure 10–5.

Figure 10–5. Example of the Logic for Pins IO0–IO3



The four LSBs of the ASPCR, bits CIO0–CIO3, are for configuring each pin as an input or an output. For example, as shown in the figure, setting CIO0 to 1 configures IO0 as an output; setting CIO0 to 0 configures IO0 as an input. At reset, CIO0–CIO3 are all cleared to 0, making all four of the the pins inputs. Table 10–5 summarizes the configuration of the pins.

Table 10–5. Configuring Pins IO0–IO3 with ASPCR Bits CIO0–CIO3

CIO0 Bit	IO0 Pin	CIO1 Bit	IO1 Pin	CIO2 Bit	IO2 Pin	CIO3 Bit	IO3 Pin
0	Input	0	Input	0	Input	0	Input
1	Output	1	Output	1	Output	1	Output

When pins IO0–IO3 are configured as inputs

When pins IO0–IO3 are configured as inputs, the eight LSBs of the IOSR allow you to monitor these four pins. Each of the IOSR bits 3–0, called IO3, IO2, IO1, and IO0, can be used to read the current logic level (high or low) of the signal at the corresponding pin. Each of the bits 7–4, called DIO3, DIO2, DIO1, and DIO0, is used to track a change from a previous known or unknown signal value at the corresponding pin. When a change is detected on one of the pins, the corresponding detect bit is set to 1, and an interrupt request is sent to the CPU on the TXRXINT interrupt line. You can clear each of the detect bits to 0 by writing a 1 to it. DIO3–DIO0 are only useful when the pins are configured as inputs and the serial port is enabled by the URST bit of the ASPCR (URST = 1). Table 10–6 summarizes what IOSR bits 0–7 indicate when IO0–IO3 are inputs.

Table 10–6. Viewing the Status of Pins IO0–IO3 With IOSR Bits IO0–IO3 and DIO0–DIO3

IOSR Bit Number	IOSR Bit Name	When IO0–IO3 are inputs, this bit indicates...
0	IO0	Current logic level (0 or 1) on pin IO0
1	IO1	Current logic level (0 or 1) on pin IO1
2	IO2	Current logic level (0 or 1) on pin IO2
3	IO3	Current logic level (0 or 1) on pin IO3
4	DIO0†	Change detected (1) or not detected (0) on pin IO0 (when IO0 is an input)
5	DIO1†	Change detected (1) or not detected (0) on pin IO1 (when IO1 is an input)
6	DIO2†	Change detected (1) or not detected (0) on pin IO2 (when IO2 is an input)
7	DIO3†	Change detected (1) or not detected (0) on pin IO3 (when IO3 is an input)

† Write a 1 to this bit to clear it to 0.

When pins IO0–IO3 are configured as outputs

When pins IO0–IO3 are configured as outputs, you can write to the four LSBs (IO3–IO0) of the IOSR. The value you write to each bit becomes the new logic level at the corresponding pin. For example, if you write a 0 to bit 2, the logic level at pin IO2 changes to low; if you write a 1 to bit 2, the logic level on IO2 changes to high.

10.3.6 Using Interrupts

The asynchronous serial port interrupt (TXRXINT) can be generated by three types of interrupts:

- ❑ Transmit interrupts. A transmit interrupt is generated when the ADTR empties during transmission. This indicates that the port is ready to accept a new transmit character. In addition to generating the interrupt, the port sets the THRE bit of the IOSR to 1. Transmit interrupts can be disabled by the TIM bit of the ASPCR.
- ❑ Receive interrupts. Any one of the following events will generate a receive interrupt:
  - ■ The ADTR holds a new character. This event is also indicated by the DR bit of the IOSR (DR = 1).

- Overrun occurs. The last character in the ADTR was not read before the next character overwrote it. Overrun also sets the OE bit of the IOSR to 1.
- A framing error occurs. The character received did not have a valid (logic 1) stop bit. This event is also indicated by the FE bit of the IOSR (FE = 1).
- A break has been detected on the RX pin. This event also sets the BI bit of the IOSR to 1.
- The character A or a has been detected in the ADTR by the auto-baud detect logic. This event also sets the ADC bit of the IOSR to 1. This interrupt will occur regardless of the values of the DIM, TIM, and RIM bits of the ASPCR.

With the exception of the A detect interrupt, receive interrupts can be disabled by the RIM bit of the ASPCR.

- Delta interrupts. This type of interrupt is generated if a change takes place on one of the I/O lines (IO0, IO1, IO2, or IO3) when the lines are used for ASP control (when DIM = 1 in the ASPCR). The event is also indicated by the corresponding detect bit (DIO0, DIO1, DIO2, or DIO3) in the IOSR. Delta interrupts can be disabled by the DIM bit of the ASPCR.

TXRXINT leads the CPU to interrupt vector location 000Ch in program memory. The branch at that location should lead to an interrupt service routine that identifies the cause of the interrupt and then acts accordingly. TXRXINT has a priority level of 9 (1 being highest).

TXRXINT is a maskable interrupt and is controlled by the interrupt mask register (IMR) and interrupt flag register (IFR).

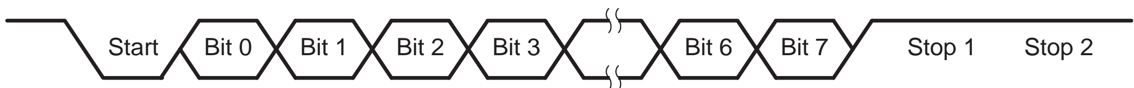
Note:

To avoid a double interrupt from the ASP, clear the IFR bit (TXRXINT) in the corresponding interrupt service routine, just before returning from the routine.

10.4 Transmitter Operation

The transmitter consists of an 8-bit transmit register (ADTR) and an 8-bit transmit shift register (AXSR). Data to be transmitted is written to the ADTR, and then the port transfers the data to the AXSR. Data written to the transmit register should be written in right-justified form, with the LSB as the rightmost bit. Data from the AXSR is shifted out on the TX pin in the serial form shown in Figure 10–6 (the number of stop bits depends on the value of the STB bit in the ASPCR). When the serial port is not transmitting, TX should be held high by clearing the SETBRK bit of the ASPCR (SETBRK = 0).

Figure 10–6. Data Transmit



Transmission is started by a write to the ADTR. If the AXSR is empty, data from the ADTR is transferred to the AXSR. If the AXSR is full, then data is kept in the ADTR, and existing data in the AXSR is shifted out to the sequence control logic. If both the AXSR and ADTR are full and the CPU tries to write to the ADTR, the write is not allowed, and existing data in both registers is maintained.

If the transmit register is empty and interrupt TXRXINT is unmasked (in the IMR) and enabled (by the INTM bit), an interrupt is generated. When the ADTR empties, the THRE bit of the IOSR is set to 1. The bit is cleared when a character is loaded into the transmit register. Bit 12 (TEMT) of the IOSR is set if both the transmit and transmit shift registers are empty.

The sequence control logic constructs the transmit frame by sending out a start bit followed by the data bits from the AXSR and either one or two stop bits.

Here is a summary of asynchronous mode transmission:

1) An interrupt (TXRXINT) is generated if the transmit register is empty.

2) If AXSR is empty, the data is transferred from ADTR to AXSR.

3) A start bit is transmitted to TX, followed by eight data bits (LSB first), and the stop bit(s).

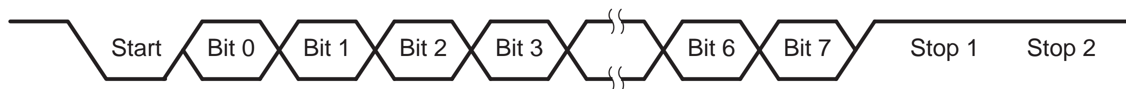
4) For the next transmission, the process begins again from step 1.

To avoid double interrupts, the interrupt service routine should clear TXRXINT in the interrupt flag register (IFR), just before forcing a return from the routine. Take special care when using this interrupt; it will be generated frequently for as long as the transmit register is empty.

10.5 Receiver Operation

The receiver includes two internal 8-bit registers: the receive register (ADTR) and receive shift register (ARSR). The data received at the RX pin should have the serial form shown in Figure 10–7 (the number of stop bits required depends on the value of the STB bit in the ASPCR).

Figure 10–7. Data Receive



Data is received on the RX pin, and the negative-edge detect logic initiates a receive operation and checks for a start bit. After the eight data bits are received, a stop bit (or bits) should be received, indicating the end of that block. If a valid stop bit is not received, a framing error has occurred; in response, the FE bit in the ASPCR is set to 1, and a TXRXINT interrupt is generated. Then normal reception continues, and the receiver looks for the next start bit.

Once a valid stop bit is received, data is then transferred to the ADTR, and an interrupt (TXRXINT) is sent to the CPU. The DR bit of the IOSR is set to indicate that a character has been received in the receive register, ADTR. (DR is cleared to 0 when the ADTR is read.) The ARSR is now available to receive another character.

If ADTR is not read before new data is transferred into the ADTR, the overflow error (OE) flag is set in the IOSR.

In summary, asynchronous mode reception involves the following events:

1) A negative edge is received on RX to indicate a start bit. A test is performed to indicate whether a start bit is valid.

2) If the start bit is valid, eight data bits are shifted into ARSR (LSB first).

3) A stop bit is received to indicate end of reception. (If a stop bit is not received, a framing error is indicated.)

4) Data is transferred from ARSR to ADTR.

5) An interrupt is sent to the CPU once data has been placed in the ADTR.

6) Reception is complete. The receiver waits for another negative transition.

To avoid double interrupts, the interrupt service routine should clear TXRXINT in the interrupt flag register (IFR) just before forcing a return from the routine.

Chapter 11

TMS320C209

All ’C20x devices use the same central processing unit (CPU), bus structure, and instruction set, but the ’C209 has some notable differences. This chapter compares features on the ’C209 with those on other ’C20x devices and then provides information specific to the ’C209 in the areas of memory and I/O spaces, interrupts, and on-chip peripherals.

Topic **Page**

11.1 ’C209 Versus Other ’C20x Devices 11-2

11.2 ’C209 Memory and I/O Spaces 11-5

11.3 ’C209 Interrupts 11-10

11.4 ’C209 On-Chip Peripherals 11-15

11.1 ’C209 Versus Other ’C20x Devices

This section explains the differences between the ’C209 and other ’C20x devices and concludes with a table to help you find the other information in this manual that applies to the ’C209.

11.1.1 What Is the Same

The following components and features are identical on all ’C20x devices, including the ’C209:

- Central processing unit
- Status registers ST0 and ST1
- Assembly language instructions
- Addressing modes
- Global data memory
- Program-address generation logic
- General-purpose I/O pins $\overline{\text{BIO}}$ and XF

11.1.2 What Is Different

The important differences between the ’C209 and other ’C20x devices are as follows:

- Peripherals**:**
  - The ’C209 has no serial ports.
  - The wait-state generator can be programmed to generate either no wait states or one wait state. Other ’C20x devices provide zero to seven wait states.
  - The wait-state generator does not provide separate wait states for the upper and lower halves of program memory.
  - The ’C209 supports address visibility mode (enabled with the wait-state generator control register). In this mode, the device passes the internal program address to the external address bus when this bus is not used for an external access.
  - The ’C209 clock generator supports only two options: multiply-by-two (×2) and divide-by-two (÷2).
  - The ’C209 does not have a CLK register; thus it cannot prevent the CLKOUT1 signal from appearing on the CLKOUT1 pin.
  - The ’C209 does not have I/O pins IO3, IO2, IO1, and IO0.

- Memory and I/O Spaces:
  - The I/O addresses of the peripheral registers are different on the 'C209.
  - The 'C209 does not support the 'C20x HOLD operation.
- Interrupts:
  - The 'C209 has four maskable interrupt lines, none of them shared. The other devices have six interrupt lines, one shared by the $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ pins.
  - The 'C209 does not have an interrupt control register (ICR) because $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ have their own interrupt lines.
  - Although the interrupt flag register (IFR) and interrupt mask register (IMR) are used in the same way on all 'C20x device, the 'C209 has fewer flag and mask bits because it does not have serial ports.
  - On the 'C209, interrupts $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ have their own interrupt lines and, thus, have their own interrupt vectors. On other 'C20x devices, $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ share an interrupt line and, thus, share one interrupt vector.
  - The 'C209 has an interrupt acknowledge pin ($\overline{\text{IACK}}$), which allows external detection of when an interrupt has been acknowledged.
  - The 'C209 has two pins for reset: $\overline{\text{RS}}$ and RS; other 'C20x devices have only $\overline{\text{RS}}$.

11.1.3 Where to Find the Information You Need About the TMS320C209

For information about:		Look here:
Assembly language instructions		Chapter 7, *Assembly Language Instructions*
Clock generator	Main description	Chapter 8, *On-Chip Peripherals*
	Options and configuration	Section 11.4.1 (page 11-15)
CPU		Chapter 3, *Central Processing Unit*
Data-address generation		Chapter 6, *Addressing Modes*
I/O Space	Main description	Chapter 4, *Memory*
	Effect of READY pin	Section 11.2 (page 11-5)
	Control register locations	Table 11–3 (page 11-9)

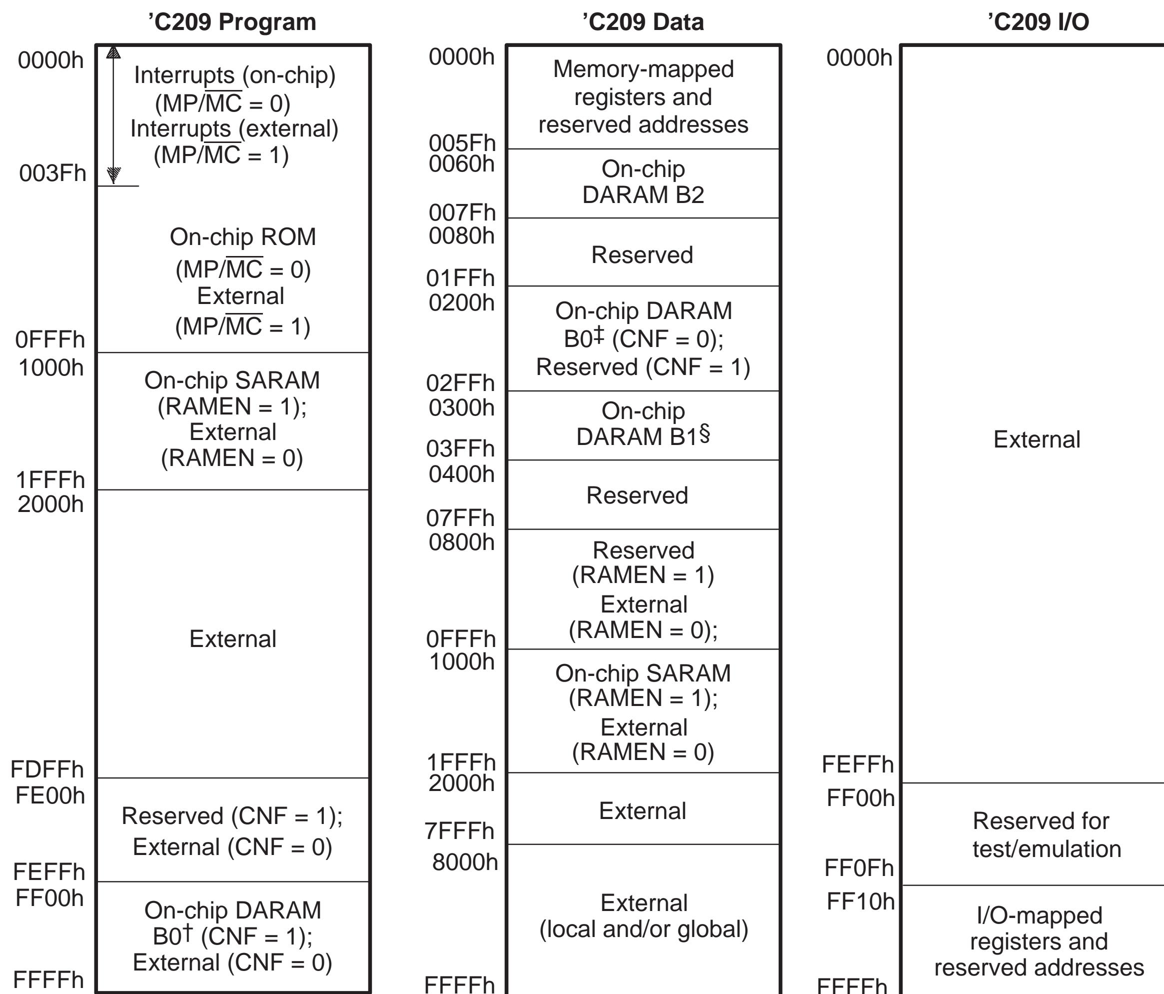
For information about:		Look here:
Interrupts	Main description	Chapter 5, *Program Control*
	Vector locations	Table 11–4 (page 11-10)
	Flag and mask registers	Section 11.3.1 (page 11-12)
	Interrupt acknowledge pin	Section 11.3.2 (page 11-14)
Memory	Main description	Chapter 4, *Memory*
	Address maps	Figure 11–1 (page 11-6)
	Configuration	Section 11.2 (page 11-5)
Pipeline		Chapter 5, *Program Control*
Power-down mode		Chapter 5, *Program Control*
Program-address generation		Chapter 5, *Program Control*
Program control		Chapter 5, *Program Control*
Stack		Chapter 5, *Program Control*
Status registers		Chapter 5, *Program Control*
Timer	Main description	Chapter 8, *On-Chip Peripherals*
	Configuration	Section 11.4.2 (page 11-16)
Wait-state generator	Main description	Chapter 8, *On-Chip Peripherals*
	Configuration	Section 11.4.3 (page 11-17)

11.2 ’C209 Memory and I/O Spaces

The ’C209 does not have an on-chip bootloader and does not support the ’C20x HOLD operation. Figure 11–1 shows the ’C209 address map. The on-chip program and data memory available on the ’C209 consists of:

- ROM (4K words, for program memory)
- SARAM (4K words, for program and/or data memory)
- DARAM B0 (256 words, for program or data memory)
- DARAM B1 (256 words, for data memory)
- DARAM B2 (32 words, for data memory)

Figure 11–1. 'C209 Address Maps



† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to here as reserved when CNF = 1.

‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to here as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to here as reserved.

CAUTION

Do Not Write to Reserved Addresses

To avoid unpredictable operation of the processor, do not write to any addresses labeled Reserved. This includes any data-memory address in the range 0000h–005Fh that is not designated for an on-chip register and any I/O address in the range FF00h–FFFFh that is not designated for an on-chip register.

You select or deselect the ROM by changing the level on the MP/$\overline{MC}$ pin at reset:

- ❑ When MP/$\overline{MC}$ = 0 (low) at reset, the device is configured as a microcomputer. The on-chip ROM is enabled and is accessible at addresses 0000h–0FFFh. The device fetches the reset vector from on-chip ROM.
- ❑ When MP/$\overline{MC}$ = 1 (high) at reset, the device is configured as a microprocessor, and addresses 0000h–0FFFh are used to access external memory. The device fetches the reset vector from external memory.

Regardless of the value of MP/$\overline{MC}$, the ’C20x fetches its reset vector at location 0000h of program memory.

The addresses assigned to the on-chip SARAM are shared by program memory and data memory. The RAMEN signal allows you to toggle the data addresses 1000h–1FFFh and the program addresses 1000h–1FFFh between on-chip memory and external memory:

- ❑ When RAMEN = 1 (high), program addresses 1000h–1FFFh and data addresses 1000h–1FFFh are mapped to the same physical locations in the on-chip SARAM. For example, 1000h in program memory and 1000h in data memory point to the same physical location in the on-chip SARAM. Thus, the 4K words of on-chip SARAM are accessible for program and/or data space.

Note:

When RAMEN = 1, program addresses 1000h–1FFFh and data addresses 1000h–1FFFh are one and the same. When writing data to these locations be careful not to overwrite existing program instructions.

- ❑ When RAMEN = 0 (low), program addresses 1000h–1FFFh (4K) are mapped to external program memory and data addresses 1000h–1FFFh

(4K) are mapped to external data memory. Thus, a total of 8K additional addresses (4K program and 4K data) are available for external memory.

DARAM blocks B1 and B2 are fixed, but DARAM block B0 may be mapped to program space or data space, depending on the value of the CNF bit (bit 12 of status register ST1):

- When CNF = 0, B0 is mapped to data space and is accessible at data addresses 0200h–02FFh. Note that the addressable external *program* memory increases by 512 words. At reset, CNF = 0.
- When CNF = 1, B0 is mapped to program space and is accessible at program addresses FF00h–FFFFh.

Table 11–1 lists the available program memory configurations for the ’C209; Table 11–2 lists the data-memory configurations. Note these facts:

- Program-memory addresses 0000h–003Fh are used for the interrupt vectors.
- Data-memory addresses 0000h–005Fh contain on-chip memory-mapped registers and reserved memory.
- Two other on-chip data-memory ranges are always reserved: 0080h–01FFh and 0400h–07FFh.

Table 11–1. ’C209 Program-Memory Configuration Options

MP/$\overline{\text{MC}}$	RAMEN	CNF	ROM (hex)	SARAM (hex)	DARAM B0 (hex)	External (hex)	Reserved (hex)
0	0	0	0000–0FFF	–	–	1000–FFFF	–
0	0	1	0000–0FFF	–	FF00–FFFF	1000–FDFF	FE00–FEFF
0	1	0	0000–0FFF	1000–1FFF	–	2000–FFFF	–
0	1	1	0000–0FFF	1000–1FFF	FF00–FFFF	2000–FDFF	FE00–FEFF
1	0	0	–	–	–	0000–FFFF	–
1	0	1	–	–	FF00–FFFF	0000–FDFF	FE00–FEFF
1	1	0	–	1000–1FFF	–	0000–0FFF 2000–FFFF	–
1	1	1	–	1000–1FFF	FF00–FFFF	0000–0FFF 2000–FDFF	FE00–FEFF

Table 11–2. ’C209 Data-Memory Configuration Options

RAMEN	CNF	DARAM B0 (hex)	DARAM B1 (hex)	DARAM B2 (hex)	SARAM (hex)	External (hex)	Reserved (hex)
0	0	0200–02FF	0300–03FF	0060–007F	–	0800–FFFF	0000–005F 0080–01FF 0400–07FF
0	1	–	0300–03FF	0060–007F	–	0800–FFFF	0000–005F 0080–02FF 0400–07FF
1	0	0200–02FF	0300–03FF	0060–007F	1000–1FFF	2000–FFFF	0000–005F 0080–01FF 0400–0FFF
1	1	–	0300–03FF	0060–007F	1000–1FFF	2000–FFFF	0000–005F 0080–02FF 0400–0FFF

A portion of the on-chip I/O space contains the control registers listed in Table 11–3.The corresponding registers on other ’C20x devices are not at the addresses shown in this table. When accessing the I/O-mapped registers on the ’C209, also keep in mind the following:

- The READY pin must be pulled high to permit reads from or writes to registers mapped to internal I/O space. This is not true for other ’C20x devices.
- The $\overline{IS}$ (I/O select) and R/$\overline{W}$ (read/write) signals are visible on their pins during reads from or writes to registers mapped to internal I/O space. On other ’C20x devices, none of the interface signals are visible during internal I/O accesses.

Table 11–3. ’C209 On-Chip Registers Mapped to I/O Space

I/O Address	Name	Description
FFFCh	TCR	Timer control register
FFFDh	PRD	Timer period register
FFFEh	TIM	Timer counter register
FFFFh	WSGR	Wait-state generator control register

Note: The corresponding registers on other ’C20x devices are not at these addresses.

11.3 ’C209 Interrupts

Table 11–4 lists the interrupts available on the ’C209 and shows their vector locations. In addition, it shows the priority of each of the hardware interrupts. Note that a device reset can be initiated in either of two ways: by driving the $\overline{\text{RS}}$ pin low or by driving the RS pin high. The K value shown for each interrupt vector location is the operand to be used with the INTR instruction if you want to force a branch to that location.

Table 11–4. ’C209 Interrupt Locations and Priorities

K†	Vector Location	Name	Priority	Function
0	0h	$\overline{\text{RS}}$ or RS‡	1 (highest)	Hardware reset (nonmaskable)
1	2h	$\overline{\text{INT1}}$	4	User-maskable interrupt #1
2	4h	$\overline{\text{INT2}}$	5	User-maskable interrupt #2
3	6h	$\overline{\text{INT3}}$	6	User-maskable interrupt #3
4	8h	TINT	7	User-maskable interrupt #4: timer interrupt
5	Ah		8	Reserved
6	Ch		9	Reserved
7	Eh		10	Reserved
8	10h	INT8	–	User-defined software interrupt
9	12h	INT9	–	User-defined software interrupt
10	14h	INT10	–	User-defined software interrupt
11	16h	INT11	–	User-defined software interrupt
12	18h	INT12	–	User-defined software interrupt
13	1Ah	INT13	–	User-defined software interrupt
14	1Ch	INT14	–	User-defined software interrupt

† The K value is the operand used in an INTR instruction that branches to the corresponding interrupt vector location.
‡ The ’C209 has two pins for triggering a hardware reset: $\overline{\text{RS}}$ and RS. If either $\overline{\text{RS}}$ is driven low or RS is driven high, the device will be reset.

Table 11–4. ’C209 Interrupt Locations and Priorities (Continued)

K†	Vector Location	Name	Priority	Function
15	1Eh	INT15	–	User-defined software interrupt
16	20h	INT16	–	User-defined software interrupt
17	22h	TRAP	–	TRAP instruction vector
18	24h	$\overline{\text{NMI}}$	3	Nonmaskable interrupt
19	26h		2	Reserved
20	28h	INT20	–	User-defined software interrupt
21	2Ah	INT21	–	User-defined software interrupt
22	2Ch	INT22	–	User-defined software interrupt
23	2Eh	INT23	–	User-defined software interrupt
24	30h	INT24	–	User-defined software interrupt
25	32h	INT25	–	User-defined software interrupt
26	34h	INT26	–	User-defined software interrupt
27	36h	INT27	–	User-defined software interrupt
28	38h	INT28	–	User-defined software interrupt
29	3Ah	INT29	–	User-defined software interrupt
30	3Ch	INT30	–	User-defined software interrupt
31	3Eh	INT31	–	User-defined software interrupt

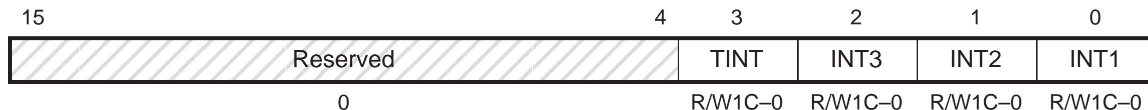
† The K value is the operand used in an INTR instruction that branches to the corresponding interrupt vector location.

‡ The ’C209 has two pins for triggering a hardware reset: $\overline{\text{RS}}$ and RS. If either $\overline{\text{RS}}$ is driven low or RS is driven high, the device will be reset.

11.3.1 ’C209 Interrupt Registers

As with other ’C20x devices, the maskable interrupts of the ’C209 are controlled by an interrupt flag register (IFR) and an interrupt mask register (IMR). Figure 11–2 shows the IFR and Figure 11–3 shows the IMR. Each of the figures is followed by descriptions of the bits.

Figure 11–2.’C209 Interrupt Flag Register (IFR) — Data-Memory Address 0006h

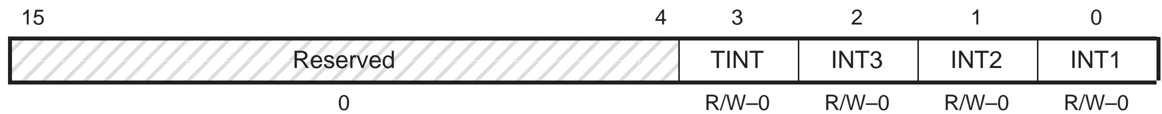


Note: 0 = Always read as zeros; R = Read access; W1C = Write 1 to this bit to clear it to 0; value following dash (–) is value after reset.

Table 11–5. ’C209 IFR — Data Memory Address 0006h Bit Descriptions

Bit No.	Name	Function
15–4	Reserved	Bits 15–4 are reserved and are always read as 0s.
3	TINT	Timer interrupt flag. Bit 3 indicates whether interrupt TINT is pending (whether TINT is requesting acknowledgment from the CPU). 0 Interrupt TINT is not pending. 1 Interrupt TINT is pending.
2	INT3	Interrupt 3 flag. Bit 2 indicates whether $\overline{\text{INT3}}$ is pending (whether $\overline{\text{INT3}}$ is requesting acknowledgment from the CPU). 0 $\overline{\text{INT3}}$ is not pending. 1 $\overline{\text{INT3}}$ is pending.
1	INT2	Interrupt 2 flag. Bit 1 indicates whether $\overline{\text{INT2}}$ is pending (whether $\overline{\text{INT2}}$ is requesting acknowledgment from the CPU). 0 $\overline{\text{INT2}}$ is not pending. 1 $\overline{\text{INT2}}$ is pending.
0	INT1	Interrupt 1 flag. Bit 0 indicates whether $\overline{\text{INT1}}$ is pending (whether $\overline{\text{INT1}}$ is requesting acknowledgment from the CPU). 0 $\overline{\text{INT1}}$ is not pending. 1 $\overline{\text{INT1}}$ is pending.

Figure 11–3. 'C209 Interrupt Mask Register (IMR) — Data-Memory Address 0004h



Note: **Note:** 0 = Always read as zeros; R = Read access; W = Write access; value following dash (–) is value after reset.

Table 11–6. 'C209 IMR — Data Memory Address 0004h Bit Descriptions

Bit No.	Name	Function
15–4	Reserved	Bits 15–4 are reserved and are always read as 0s.
3	TINT	Timer interrupt mask. Mask or unmask the internal timer interrupt, TINT, with this bit. 0 TINT is masked. 1 TINT is unmasked.
2	INT3	Interrupt 3 mask. Unmask external interrupt $\overline{\text{INT3}}$ by writing a 1 to this bit. 0 $\overline{\text{INT3}}$ is masked. 1 $\overline{\text{INT3}}$ is unmasked.
1	INT2	Interrupt 2 mask. Unmask external interrupt $\overline{\text{INT2}}$ by writing a 1 to this bit. 0 $\overline{\text{INT2}}$ is masked. 1 $\overline{\text{INT2}}$ is unmasked.
0	INT1	Interrupt 1 mask. Unmask external interrupt $\overline{\text{INT1}}$ by writing a 1 to this bit. 0 $\overline{\text{INT1}}$ is masked. 1 $\overline{\text{INT1}}$ is unmasked.

11.3.2 $\overline{\text{IACK}}$ Pin

On the ’C209, the interrupt acknowledge signal is available at the external $\overline{\text{IACK}}$ pin. The CPU generates this signal while it fetches the first word of any of the interrupt vectors, whether the interrupt was requested by hardware or by software. $\overline{\text{IACK}}$ is not affected by wait states; $\overline{\text{IACK}}$ goes low only on the first cycle of the read when wait states are used. At reset, the interrupt acknowledge signal is generated in the same manner as for a maskable interrupt.

Your external hardware can use the $\overline{\text{IACK}}$ signal to determine when the processor acknowledges an interrupt. Additionally, when $\overline{\text{IACK}}$ goes low, the hardware can sample the address pins (A15–A0) to determine which interrupt the processor is acknowledging. Since the interrupt vectors are spaced apart by two words, address pins A1–A4 can be decoded at the falling edge of $\overline{\text{IACK}}$ to identify the interrupt being acknowledged.

11.4 ’C209 On-Chip Peripherals

The ’C209 has these on-chip peripherals:

- Clock generator. The clock generator is fundamentally the same on all ’C20x devices, including the ’C209. However, the ’C209 is limited to the two clock modes described in section 11.4.1.
- Timer. The timer is also fundamentally the same. The difference here is that the timer control register (TCR) on the ’C209 does not offer bits for configuring timer emulation modes. Section 11.4.2 describes the ’C209 TCR.
- Wait-state generator. The wait-state generators of the ’C20x devices operate similarly; however, the ’C209 wait-state generator is different from that of other ’C20x devices in these ways:
  - It offers zero or one wait states (not zero to seven).
  - It cannot produce separate wait states for the lower (0000h–7FFFh) and upper (8000h–FFFFh) halves of program space.
  - It provides a bit for enabling or disabling address visibility mode. In this mode (not available on other ’C20x devices), the ’C209 passes the internal program address to the external address bus when this bus is not used for an external access.

  The ’C209 generator is programmable by way of the ’C209 wait-state generator control register (WSGR) and is described section 11.4.3.

11.4.1 ’C209 Clock Generator Options

The ’C209 includes two clock modes: divide-by-2 (÷2) and multiply-by-2 (×2). The ÷2 mode operates the CPU at half the input clock rate. The ×2 option doubles the input clock and phase-locks the output clock with the input clock. To enable the ÷2 mode, tie the CLKMOD pin low. To enable the ×2 mode, tie CLKMOD high. For each clock mode, Table 11–7 shows the generated CPU clock rate and shows the state of CLKMOD, the internal oscillator, and the internal phase lock loop (PLL).

Notes:

- Change CLKMOD only while the reset signal ($\overline{RS}$ or RS) is active.
- The PLL requires approximately 2200 cycles to lock the output clock signal to the input clock signal. When setting the ×2 mode, keep the reset ($\overline{RS}$ or RS) signal active until at least three cycles after the PLL has stabilized.

Table 11–7. ’C209 Input Clock Modes

Clock Mode	CLKOUT1 Rate	CLKMOD	Oscillator	PLL
÷ 2	CLKOUT1 = CLKIN ÷ 2	0	Enabled	Disabled
× 2	CLKOUT1 = CLKIN × 2	1	Disabled	Enabled

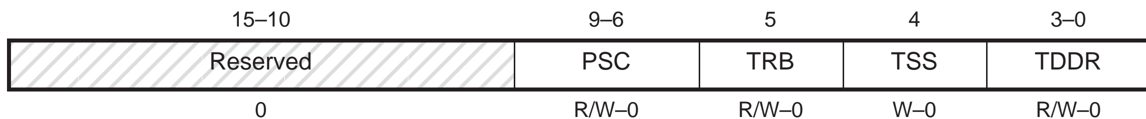
Remember the following points when configuring the clock mode:

- The modes cannot be configured dynamically. After you change the level on CLKMOD, the mode is not changed until a hardware reset is executed ($\overline{RS}$ low or RS high).
- The clock doubler mode uses an internal phase-locked loop (PLL) that requires approximately 2200 cycles to lock. Delay the rising edge of $\overline{RS}$ (or the falling edge of RS) until at least three cycles after the PLL has stabilized. When the PLL is used, the duty cycle of the CLKIN signal is more flexible, but the minimum duty cycle should not be less than 10 nanoseconds. When the PLL is not used, no phase-locking time is necessary, but the minimum pulse width must be 45% of the minimum clock cycle.

11.4.2 ’C209 Timer Control Register (TCR)

Figure 11–4 shows the bit fields of the ’C209 TCR, and descriptions of the bit fields follow the figure.

Figure 11–4.’C209 Timer Control Register (TCR) — I/O Address FFFCh



Note: 0 = Always read as zeros; R = Read access; W = Write access; value following dash (–) is value after reset.

Table 11–8. ’C209 TCR — I/O Address FFFCh Bit Descriptions

Bit No.	Name	Function
15–10	Reserved	TCR bits 10–15 are reserved and are always read as 0s.
9–6	PSC	Timer prescaler counter. These four bits hold the current prescale count for the timer. For every CLKOUT1 cycle that the PSC value is greater than 0, the PSC decrements by one. One CLKOUT1 cycle after the PSC reaches 0, the PSC is loaded with the contents of the TDDR, and the timer counter register (TIM) decrements by one. The PSC is also reloaded whenever the timer reload bit (TRB) is set by software. The PSC can be checked by reading the TCR, but it cannot be set directly. It must get its value from the timer divide-down register (TDDR). At reset, the PSC is set to 0.

Table 11–8. ’C209 TCR — I/O Address FFFCh Bit Descriptions (Continued)

Bit No.	Name	Function
5	TRB	Timer reload bit. When you write a 1 to TRB, the TIM is loaded with the value in the PRD, and the prescaler counter (PSC) is loaded with the value in the timer divide-down register (TDDR). The TRB bit is always read as zero.
4	TSS	Timer stop status bit. TSS is a 1-bit flag that stops or starts the timer. To stop the timer, set TSS to 1. To start or restart the timer, set TSS to 0. At reset, TSS is cleared to 0 and the timer immediately starts.
3–0	TDDR	Timer divide-down register. Every (TDDR + 1) CLKOUT1 cycles, the timer counter register (TIM) decrements by one. At reset, the TDDR bits are cleared to 0. If you want to increase the overall timer count by an integer factor, write this factor minus one to the four TDDR bits. When the prescaler counter (PSC) value is 0, one CLKOUT1 cycle later, the contents of the TDDR reload the PSC, and the TIM decrements by 1. TDDR also reloads the PSC whenever the timer reload bit (TRB) is set by software.

11.4.3 ’C209 Wait-State Generator

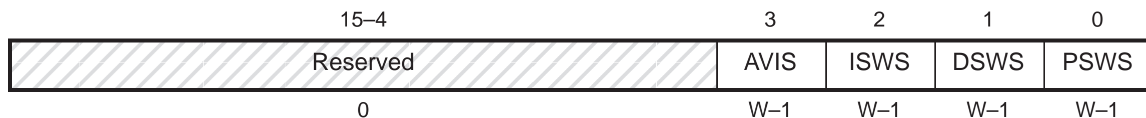
As with other ’C20x devices, the ’C209 offers two options for generating wait states:

- ❑ The READY signal. With the READY signal, you can externally generate any number of wait states.
- ❑ The on-chip wait-state generator. With the ’C209 wait-state generator, you can internally generate zero or one wait state.

The ’C209 wait-state generator inserts a wait state to a given memory space (data, program, or I/O) if the corresponding bit in WSGR is set to 1, regardless of the condition of the READY signal. As with other ’C20x devices, the READY signal can then be used to further extend wait states. The WSGR control bits are all set to 1 by reset, so that the device can operate from slow memory after reset. To avoid bus conflicts, writes from the ’C209 always take two CLKOUT1 cycles each.

To control the wait-state generator, you read from or write to the wait-state generator control register (WSGR), mapped to I/O memory location FFFFh. Figure 11–5 shows the register’s bit layout, and descriptions of the bits follow. The WSGR also enables or disables address visibility mode.

Figure 11–5.’C209 Wait-State Generator Control Register (WSGR) — I/O Address FFFFh



Note: 0 = Always read as zeros; W = Write access; value following dash (–) is value after reset.

Table 11–9. ’C209 WSGR — I/O Address FFFFh Bit Descriptions

Bit No.	Name	Function
15–4	Reserved	Bits 15–4 are reserved and are always read as 0s.
3	AVIS	Address visibility mode. AVIS = 1 enables the address visibility mode of the device. In this mode, the device provides a method of tracing internal code operation: it passes the internal program address to the address bus when this bus is not used for an external access. At reset, AVIS is set to 1. For production systems, the AVIS bit should be cleared to 0 to reduce power and noise. (AVIS does not generate a wait state.)
2	ISWS	I/O-space wait-state bit. When ISWS = 1, one wait state will be applied to all reads from off-chip I/O space. When ISWS = 0, no wait states are generated for off-chip I/O space. At reset, this bit is set to 1.
1	DSWS	Data-space wait-state bit. When DSWS = 1, one wait state will be applied to all reads from off-chip data space. When DSWS = 0, no wait states are generated for off-chip data space. At reset, this bit is set to 1.
0	PSWS	Program-space wait-state bit. When PSWS = 1, one wait state will be applied to all reads from off-chip program space. When PSWS = 0, no wait states are generated for off-chip program space. At reset, this bit is set to 1.

Appendix A

Register Summary

For the status and control registers of the ’C20x devices, this appendix summarizes:

- Their addresses
- Their reset values
- The functions of their bits

Topic **Page**

A.1 Addresses and Reset Values A-2

A.2 Register Descriptions A-4

A.1 Addresses and Reset Values

The following tables list the ’C20x registers, the addresses at which they can be accessed, and their reset values. Note that the registers mapped to internal I/O space on the ’C209 are at addresses different from those of other ’C20x devices. In addition, the ’C209 wait-state generator control register has a different reset value because there are only four control bits in the register.

Table A–1. Reset Values of the Status Registers

Name	Reset Value (Binary)	Description
ST0	XXX0 X11X XXXX XXXX	Status register 0
ST1	XXX0 X111 1111 1100	Status register 1

Notes: 1) No addresses are given for ST0 and ST1 because they can be accessed only by the CLRC, SETC, LST, and SST instructions.
2) X: Reset does not affect these bits.

Table A–2. Addresses and Reset Values of On-Chip Registers Mapped to Data Space

Name	Data-Memory Address	Reset Value	Description
IMR	0004h	0000h	Interrupt mask register
GREG	0005h	0000h	Global memory allocation register
IFR	0006h	0000h	Interrupt flag register

Table A–3. Addresses and Reset Values of On-Chip Registers Mapped to I/O Space

	I/O Address			
Name	’C209	Other ’C20x	Reset Value	Description
CLK	–	FFE8h	0000h	CLKOUT1-pin control (CLK) register
ICR	–	FFECh	0000h	Interrupt control register
SDTR	–	FFF0h	xxxxh	Synchronous data transmit and receive register
SSPCR	–	FFF1h	0030h	Synchronous serial port control register
ADTR	–	FFF4h	xxxxh	Asynchronous data transmit and receive register
ASPCR	–	FFF5h	0000h	Asynchronous serial port control register
IOSR	–	FFF6h	18xxh	I/O status register

Note: An x in the reset value represents one to four bits that are either not affected by reset or dependent on pin levels at reset.

Table A–3. Addresses and Reset Values of On-Chip Registers Mapped to I/O Space (Continued)

Name	I/O Address		Reset Value	Description
	’C209	Other ’C20x		
BRD	–	FFF7h	0001h	Baud-rate divisor register
TCR	FFFCh	FFF8h	0000h	Timer control register
PRD	FFFDh	FFF9h	FFFFh	Timer period register
TIM	FFFEh	FFFAh	FFFFh	Timer counter register
WSGR	FFFFh	FFFCh	0FFFh	Wait-state generator control register

Note: An x in the reset value represents one to four bits that are either not affected by reset or dependent on pin levels at reset.

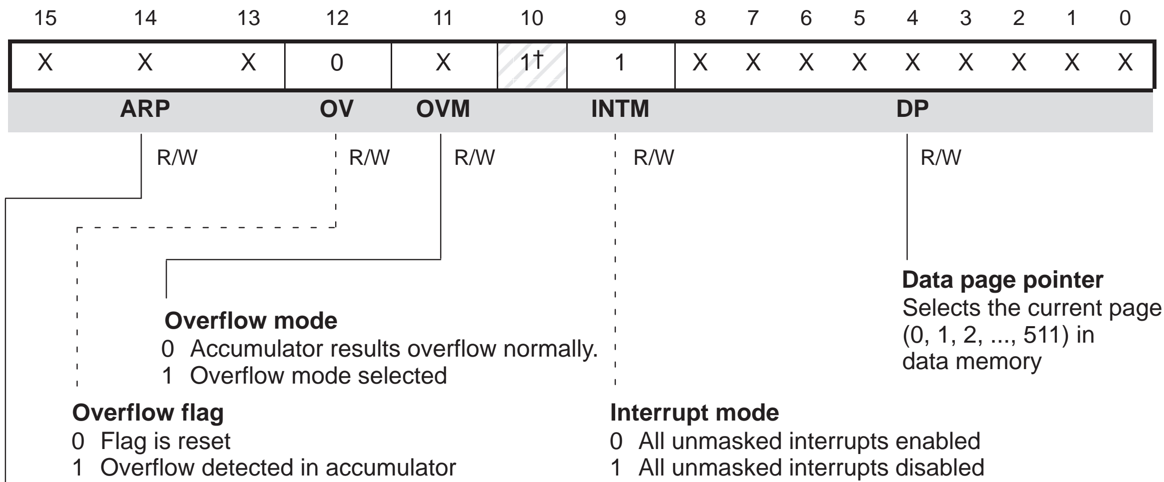
A.2 Register Descriptions

The following figures summarize the content of the ’C20x status and control registers that are divided into fields. (The other registers contain no control bits; they simply hold a single 16-bit value.) Each figure in this section provides information in this way:

- The value shown in the register is the value after reset. If the value of a particular bit is not affected by reset or depends on pin levels at reset, that bit will contain an X.
- Each unreserved bit field or set of bits has a callout that very briefly describes its effect on the processor.
- Each non-reserved bit field or set of bits is labeled with one or more of the following symbols:
  - R indicates that your software can read the bit field but cannot write to it.
  - W indicates that your software can read the bit field and write to it.
  - W1C indicates that writing a 1 to the bit field clears it to 0; writing a 0 has no effect.

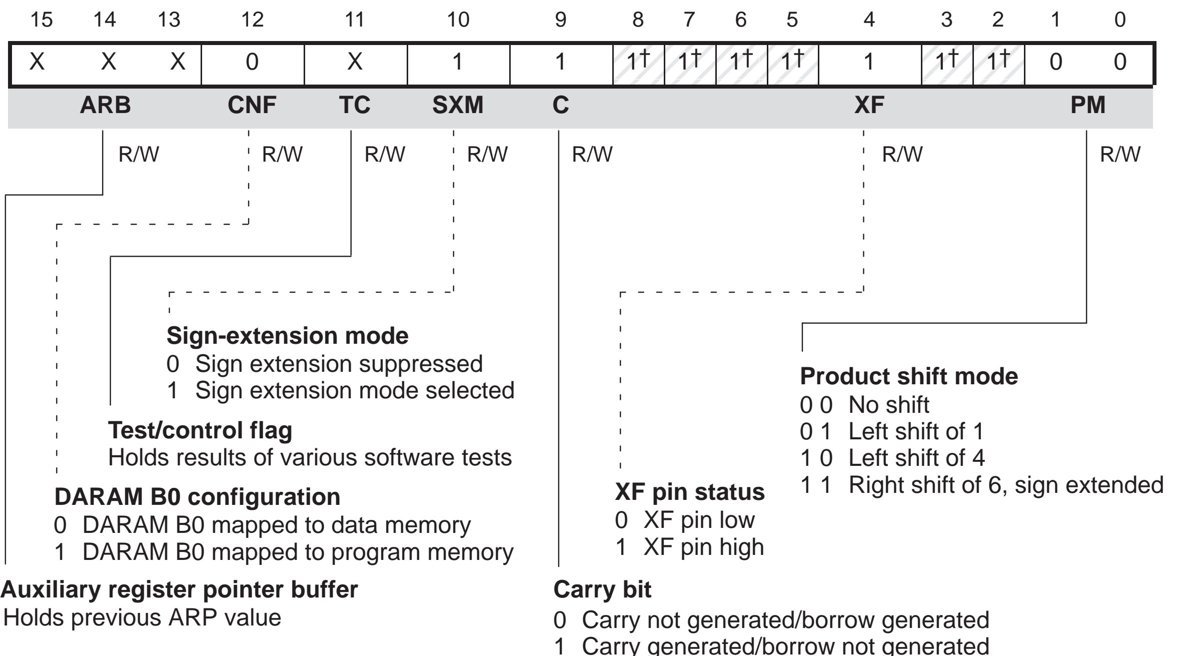
  When both read access and write access apply to a bit field, two of these symbols are shown, separated by / (a forward slash): R/W or R/W1C.
- Where needed, footnotes provide additional information for a particular figure.

Status Register ST0



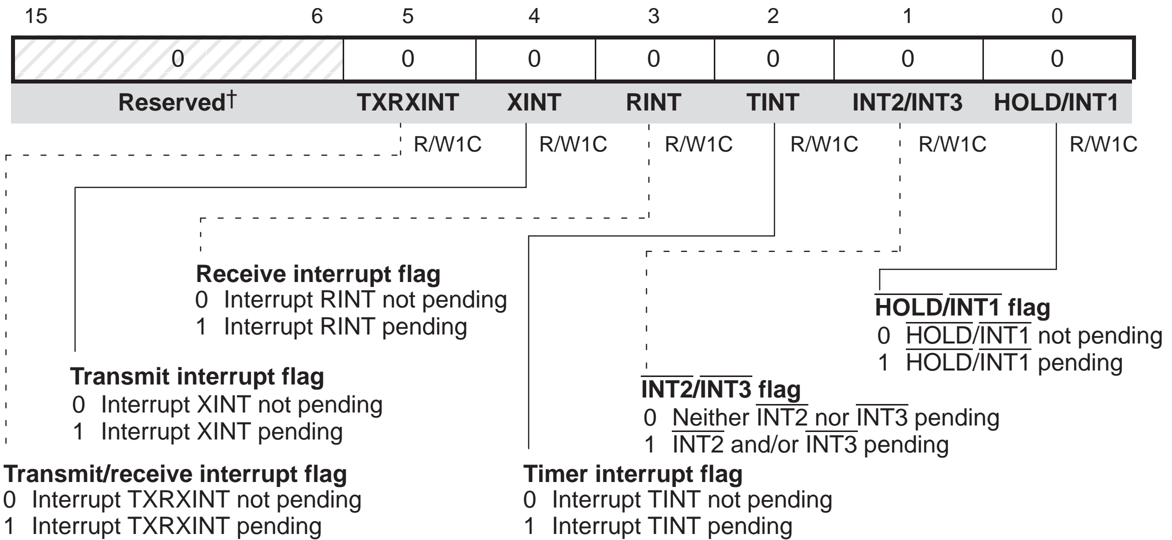
† This reserved bit is always read as 1. Writes have no effect.

Status Register ST1



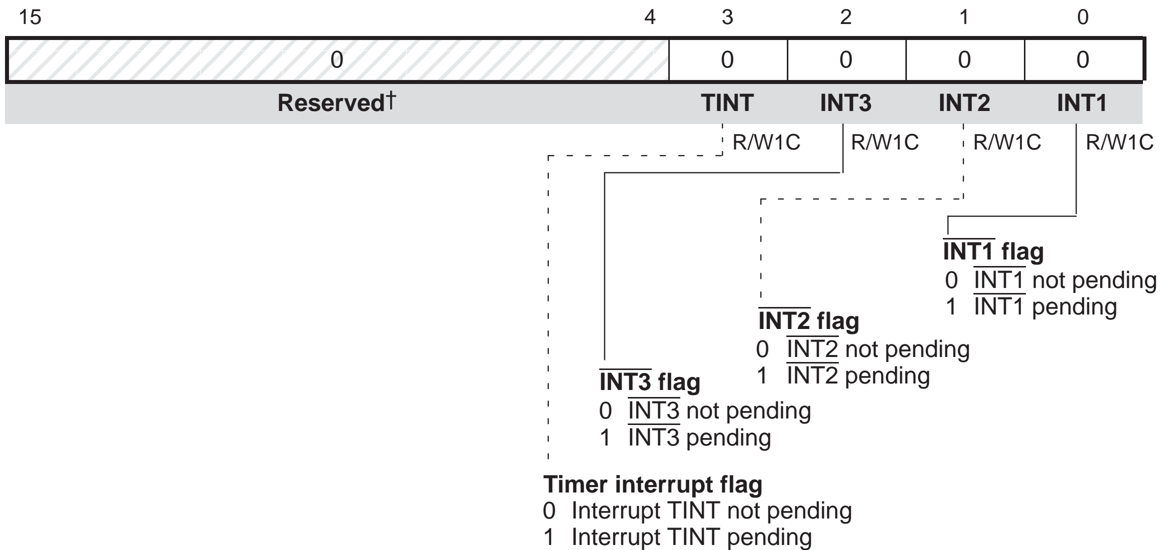
† These reserved bits are always read as 1s. Writes have no effect.

’C20x Interrupt Flag Register (IFR) — Except ’C209 — Data-Memory Address 0006h



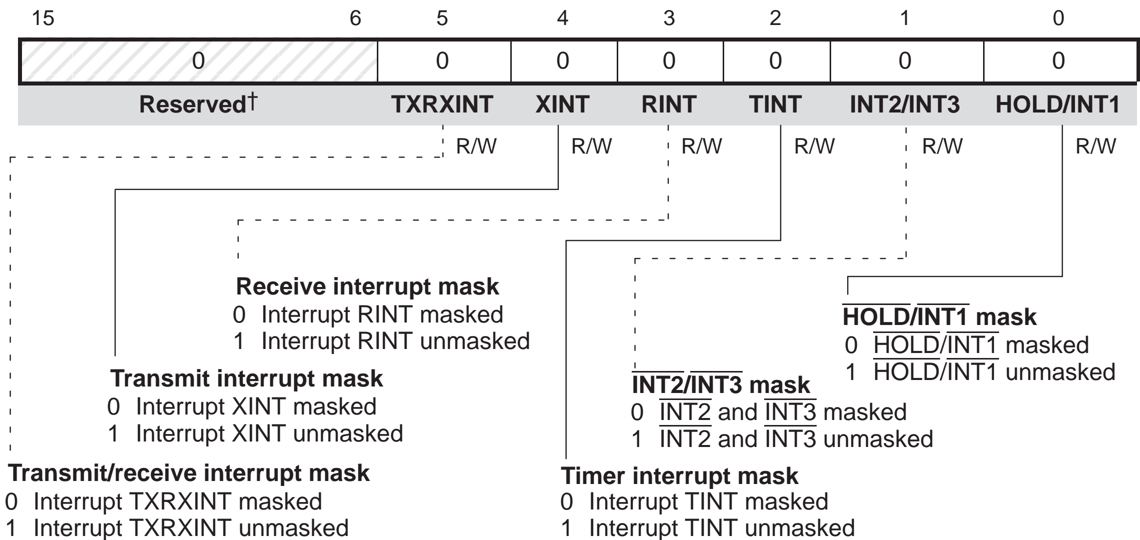
† These reserved bits are always read as 0s. Writes have no effect.

Interrupt Flag Register (IFR) — ’C209 — Data-Memory Address 0006h



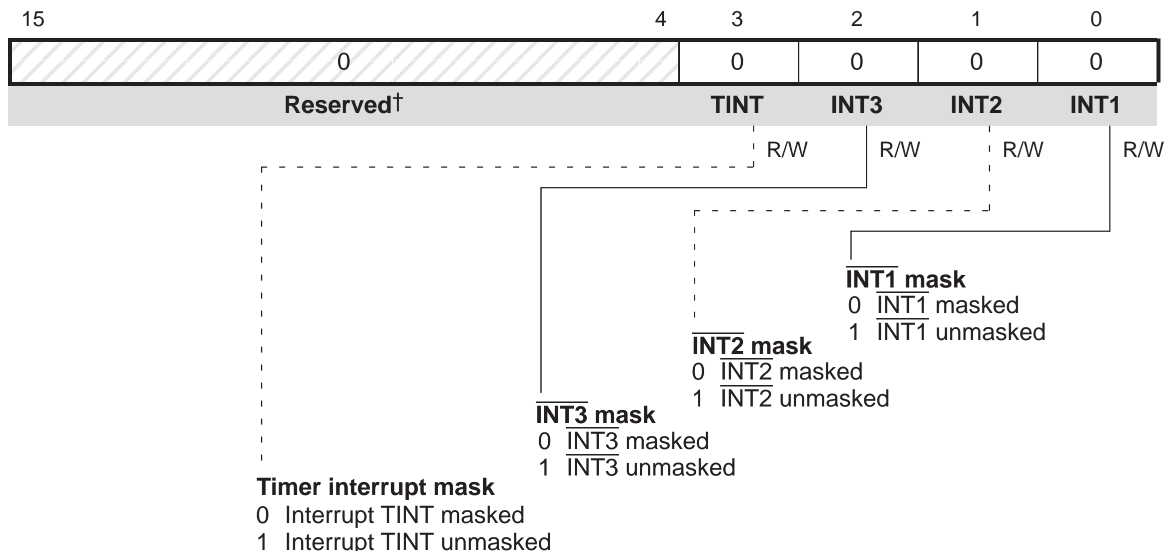
† These reserved bits are always read as 0s. Writes have no effect.

Interrupt Mask Register (IMR) — Except ’C209 — Data-Memory Address 0004h



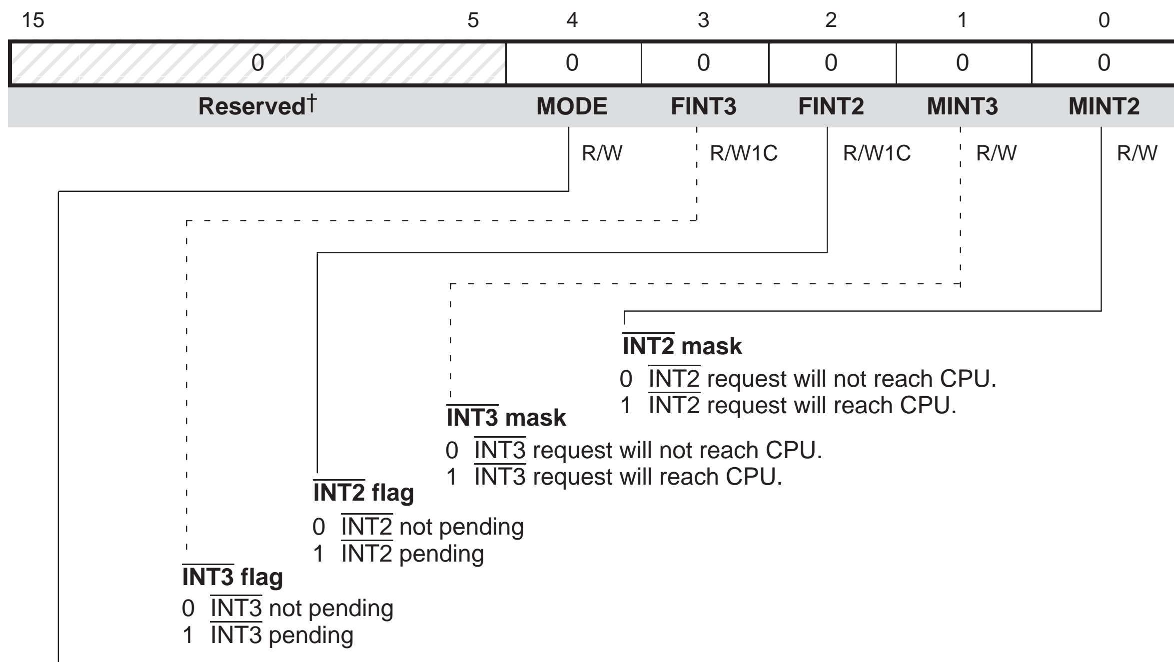
† These reserved bits are always read as 0s. Writes have no effect.

Interrupt Mask Register (IMR) — ’C209 — Data-Memory Address 0004h



† These reserved bits are always read as 0s. Writes have no effect.

Interrupt Control Register (ICR) — I/O Address FFECh

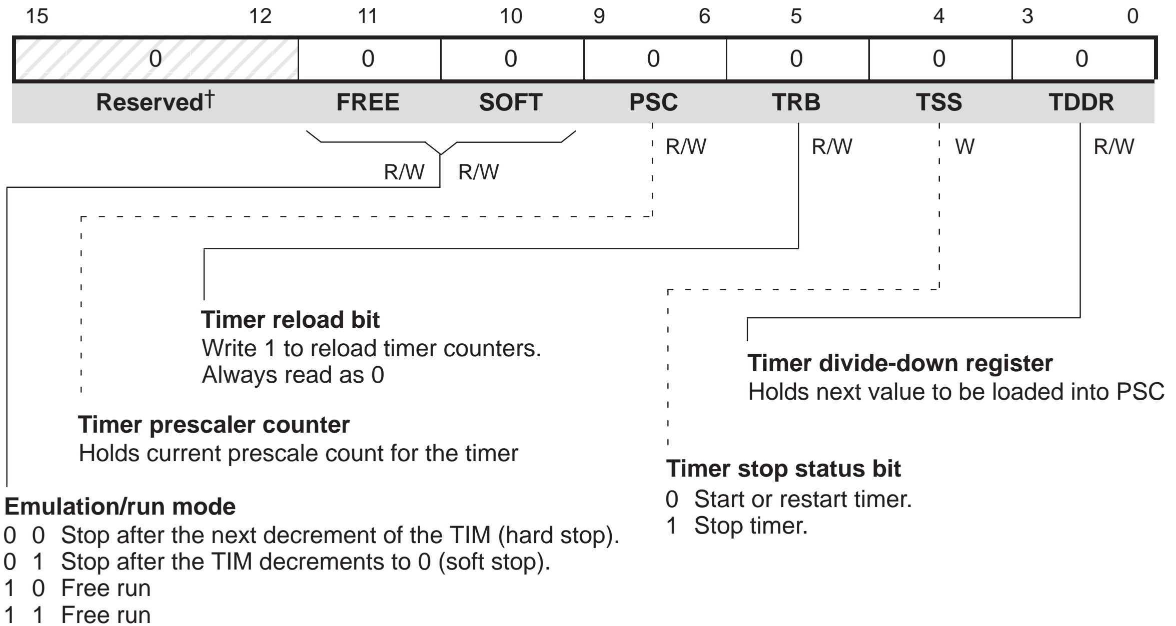


$\overline{HOLD}$/$\overline{INT1}$ pin mode

0 *Double-edge mode.* $\overline{HOLD}$/$\overline{INT1}$ pin both negative- and positive-edge sensitive
1 *Single-edge mode.* $\overline{HOLD}$/$\overline{INT1}$ pin only negative-edge sensitive

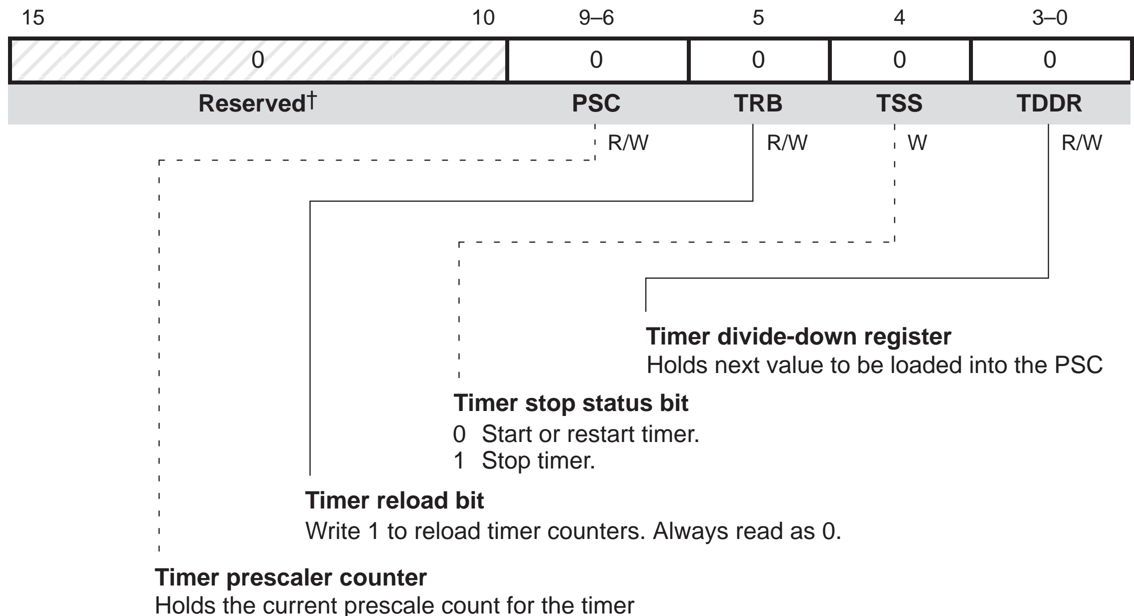
† These reserved bits are always read as 0s. Writes have no effect.

Timer Control Register (TCR) — Except ’C209 — I/O Address FFF8h



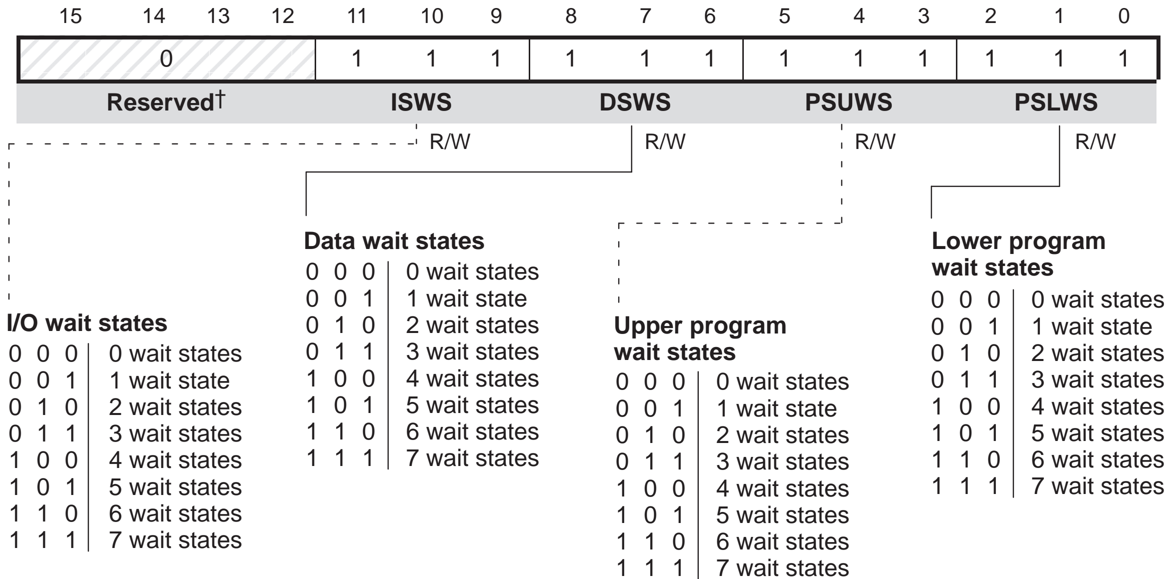
† These reserved bits are always read as 0s. Writes have no effect.

Timer Control Register (TCR) — ’C209 — I/O Address FFFCh



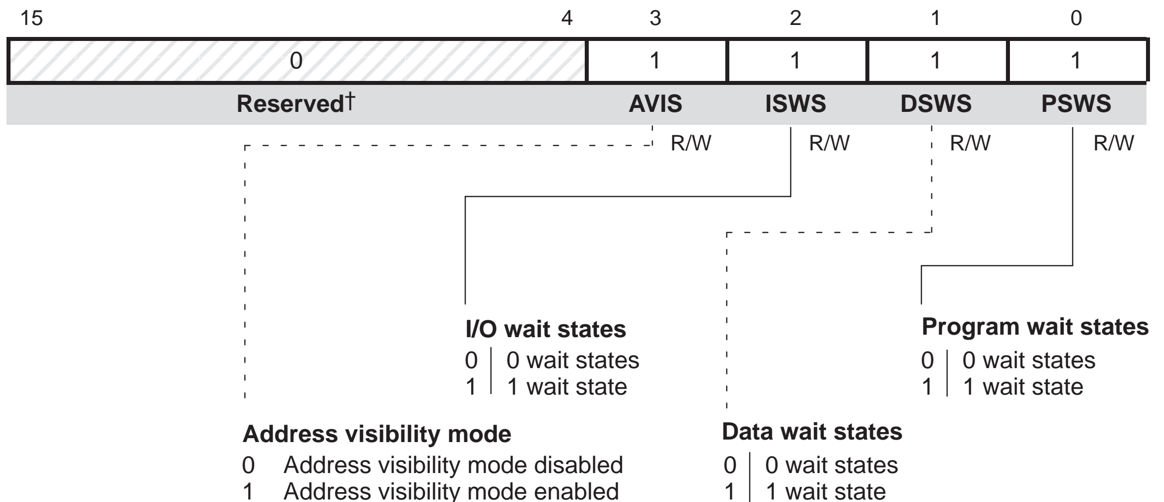
† These reserved bits are always read as 0s. Writes have no effect.

Wait-State Generator Control Register (WSGR) — Except ’C209— I/O Address FFFCh



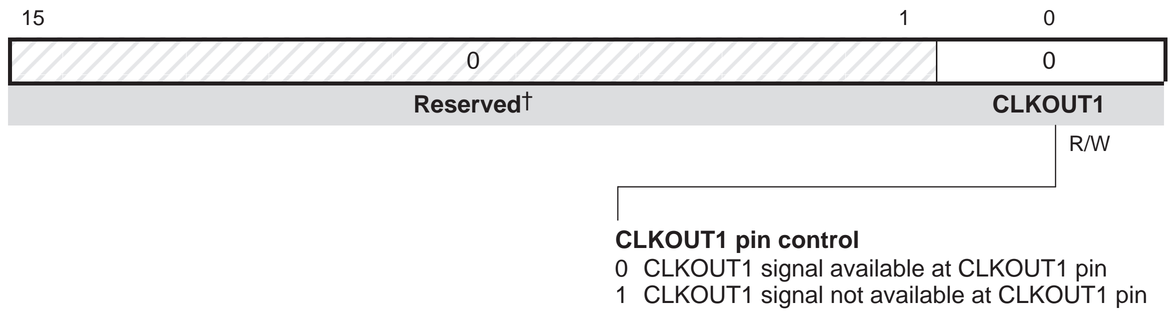
† These reserved bits are always read as 0s. Writes have no effect.

Wait-State Generator Control Register (WSGR) — ’C209 — I/O Address FFFFh



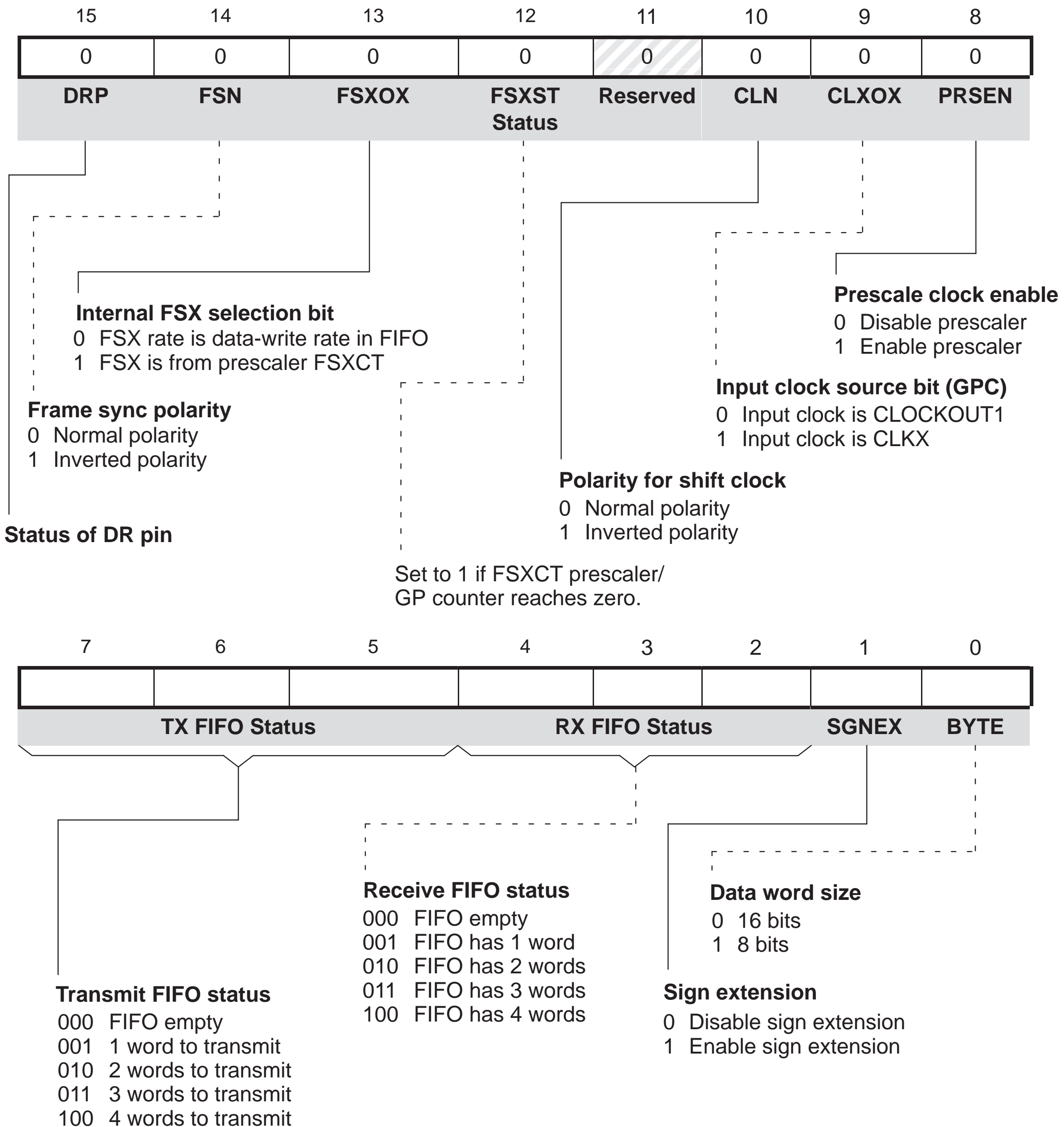
† These reserved bits are always read as 0s. Writes have no effect.

CLK Register — I/O Address FFE8h

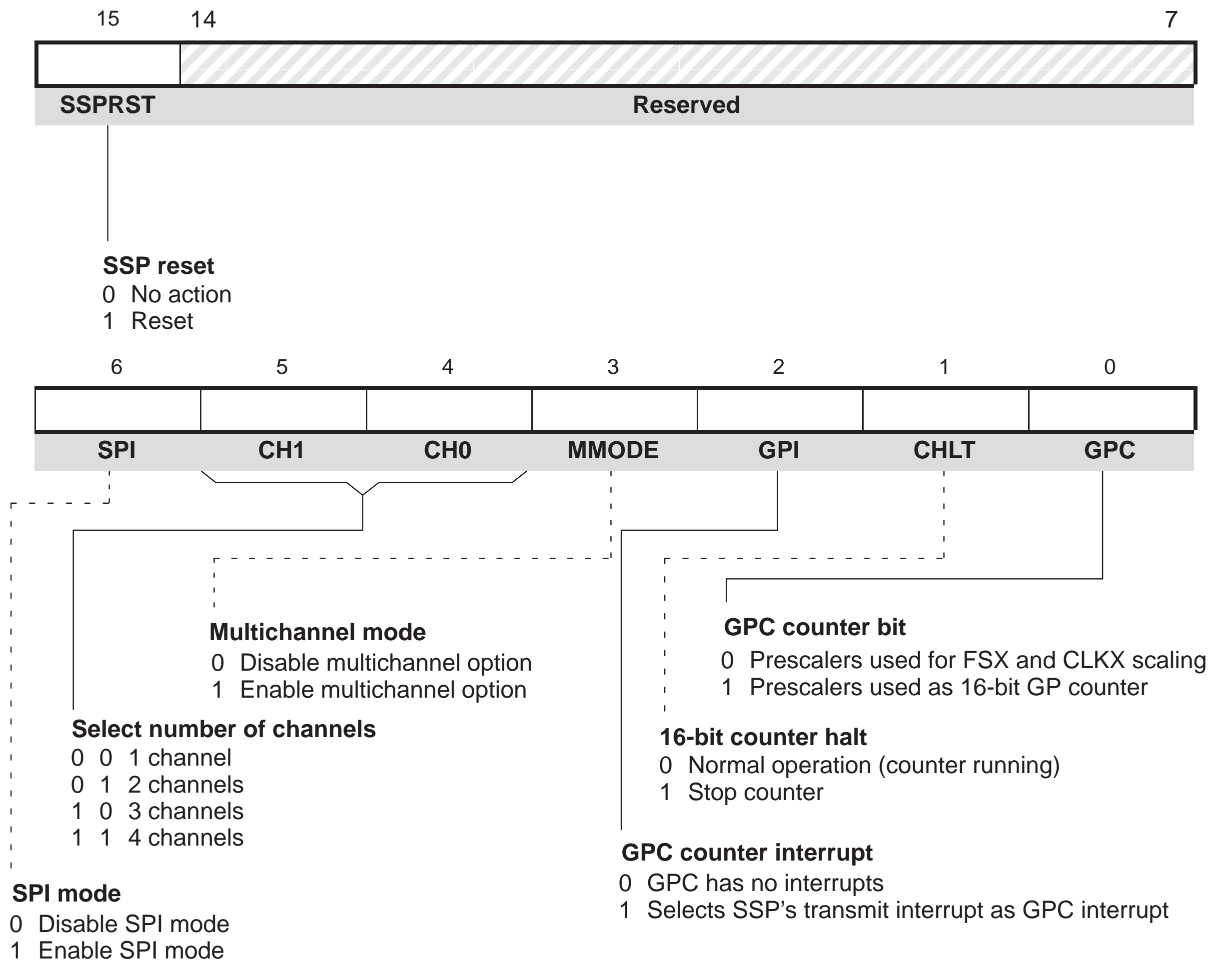


† These reserved bits are always read as 0s. Writes have no effect.

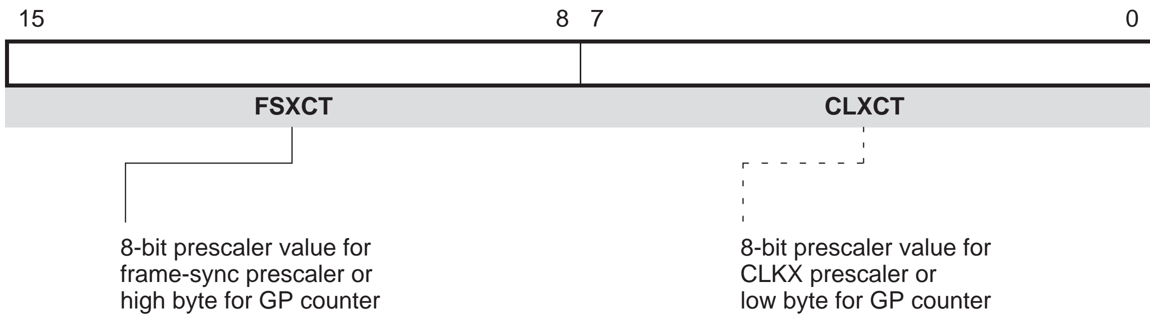
Synchronous Serial Port Status Register (SSPST) — I/O Address FFF2h



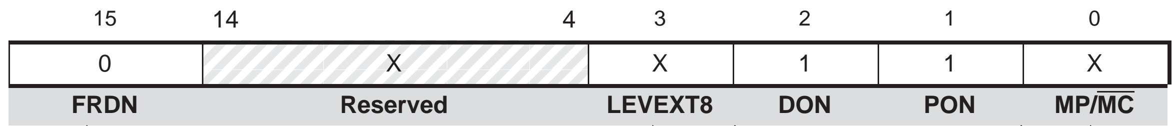
Synchronous Serial Port Multichannel Control Register (SSPMC) — I/O Address FFF3h



Synchronous Serial Port Counter Register (SSPCT) — I/O Address FFFBh



Program Memory Status Register (PMST) — I/O Address FFE4h



Latches the level of
the EXT8 pin at reset

Microprocessor/Microcomputer
0 Microcomputer
1 Microprocessor

Fast read enable
0 Use $\overline{RD}$ as read
1 Use inverted R/$\overline{W}$ as read

SARAM mapping
0 0 SARAM not mapped
0 1 SARAM in PM at 8000h
1 0 SARAM in DM at 800h
1 1 SARAM in PM and DM

Synchronous Serial Port Control Register (SSPCR) — I/O Address FFF1h

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
FREE	**SOFT**	**TCOMP**	**RFNE**	**FT1**	**FT0**	**FR1**	**FR0**
R/W	R/W	R	R	R/W	R/W	R/W	R/W

Receive FIFO buffer status
0 Receive buffer empty.
1 Receive buffer holds data.

Transmit FIFO buffer status
0 Transmit buffer empty.
1 Transmit buffer not empty.

Emulation/run mode
0 0 Immediate stop
0 1 Stop after completion of word
1 0 Free run
1 1 Free run

Generate RINT when . . .
0 0 Receive buffer not empty.
0 1 Receive buffer holds 2 or more words.
1 0 Receive buffer holds 3 or 4 words.
1 1 Receive buffer full.

Generate XINT when . . .
0 0 Transmit buffer can accept 1 or more words.
0 1 Transmit buffer can accept 2 or more words.
1 0 Transmit buffer can accept 3 or 4 words.
1 1 Transmit buffer empty (can accept 4 words).

7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0
OVF	**IN0**	**XRST**	**RRST**	**TXM**	**MCM**	**FSM**	**DLB**
R	R	R/W	R/W	R/W	R/W	R/W	R/W

Receiver reset
0 Receiver in reset
1 Receiver enabled

Transmitter reset
0 Transmitter in reset
1 Transmitter enabled

CLKR pin status
0 Level on CLKR pin is low.
1 Level on CLKR pin is high.

Overflow flag
0 No overflow condition
1 Overflow detected in receive buffer

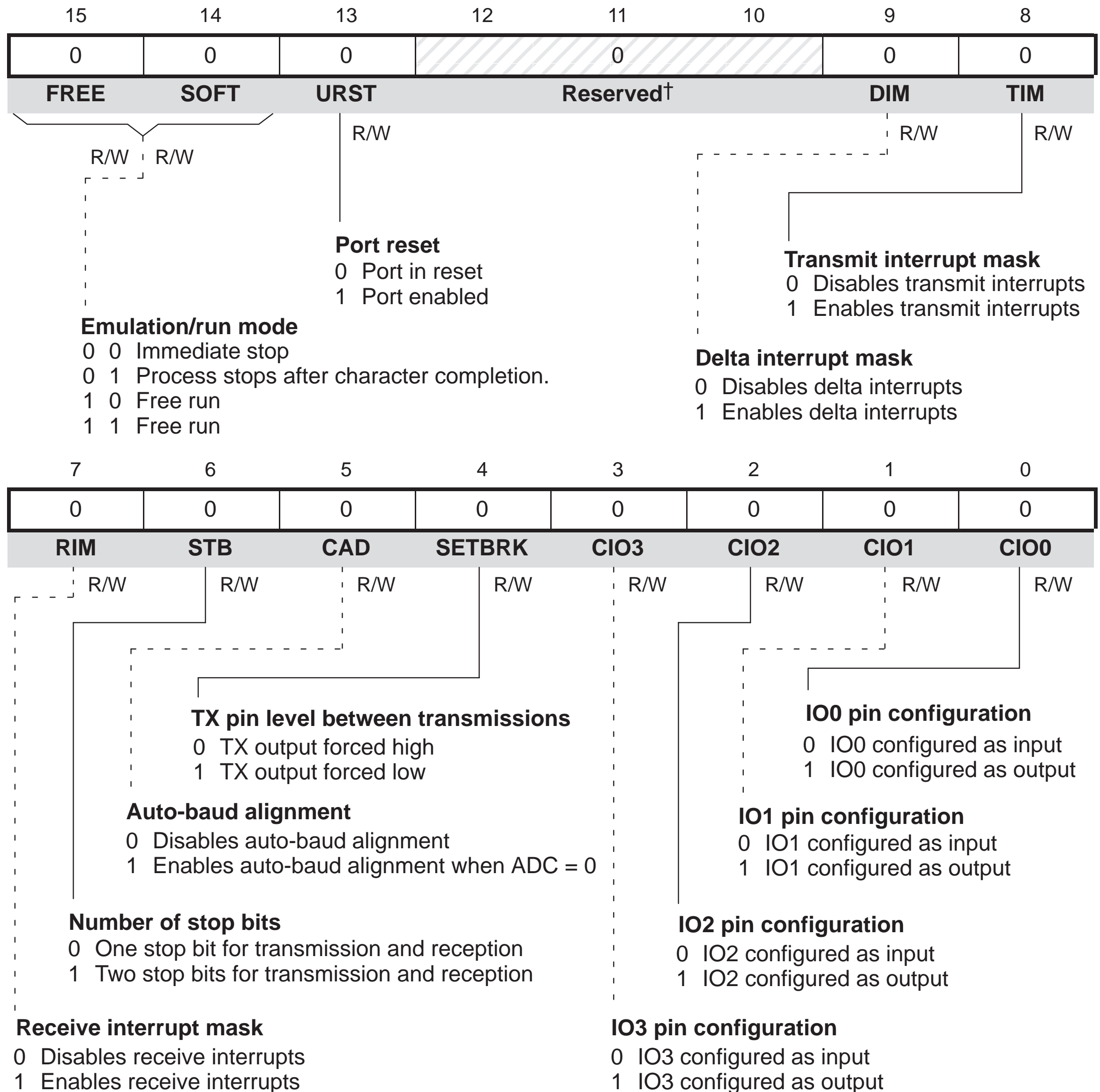
Digital loopback mode
0 Digital loopback mode disabled
1 Digital loopback mode enabled

Frame sync mode
0 Continuous mode
1 Burst mode

Transmit clock source
0 External clock source
1 Internal clock source

Transmit frame sync source
0 External frame sync source
1 Internal frame sync source

Asynchronous Serial Port Control Register (ASPCR) — I/O Address FFF5h



† These reserved bits are always read as 0s. Writes have no effect.

I/O Status Register (IOSR) — I/O Address FFF6h

15	14	13	12	11	10	9	8
0	0	0	1	1	0	0	0
Reserved†	**ADC‡**	**BI‡**	**TEMT**	**THRE‡**	**FE‡**	**OE‡**	**DR‡**
	R/W1C	R/W1C	R	R	R/W1C	R/W1C	R

Transmit empty indicator
0 ADTR and/or AXSR are full.
1 ADTR and AXSR are empty; ADTR is ready for a new character to transmit.

Break interrupt indicator
0 Normal operation
1 Break has been detected on RX pin.

***A* detect complete bit**
0 Normal operation.
1 CAD bit of ASPCR is 1 and *A* or *a* is received in ADTR.

Data ready indicator for receiver
0 Receive register empty
1 Character has been completely received.

Receive register overrun indicator
0 No overrun error detected.
1 Last character in ADTR was not read before the next character overwrote it.

Framing error indicator
0 No framing error detected.
1 Character received did not have a valid stop bit.

Transmit register empty indicator
0 Transmit register not empty. Port operation normal.
1 Transmit register empty. Port ready to receive new character.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X
DIO3‡	**DIO2‡**	**DIO1‡**	**DIO0‡**	**IO3§**	**IO2§**	**IO1§**	**IO0§**
R/W1C	R/W1C	R/W1C	R/W1C	R/W	R/W	R/W	R/W

Change detect bit for IO0
0 No change detected on IO0
1 Change detected on IO0

Change detect bit for IO1
0 No change detected on IO1
1 Change detected on IO1

Change detect bit for IO2
0 No change detected on IO2
1 Change detected on IO2

Change detect bit for IO3
0 No change detected on IO3
1 Change detected on IO3

IO0 pin status
0 IO0 signal low
1 IO0 signal high

IO1 pin status
0 IO1 signal low
1 IO1 signal high

IO2 pin status
0 IO2 signal low
1 IO2 signal high

IO3 pin status
0 IO3 signal low
1 IO3 signal high

† This reserved bit is always read as 0. Writes have no effect.
‡ When any one of these bits changes in response to the specified event, an interrupt request is generated on the TXRXINT line.
§ This bit can be written to only when the corresponding pin is configured (in the ASPCR) as an output.

Appendix B

TMS320F206 Flash Serial Loader

The TMS320F206 devices are shipped with a serial bootloader code in the flash 0 array. This appendix explains the memory map, serial port connections, and a level 1 flow chart for the ’F206 serial loader. There is also a functional description section that contains information regarding software modules, operation, and host utility loading status/modes for the ’F206.

Topic **Page**

B.1 TMS320F206 Flash Serial Loader Features B-2

B.2 Functional Description B-3

B.3 Serial Loader Code B-6

B.1 TMS320F206 Flash Serial Loader Features

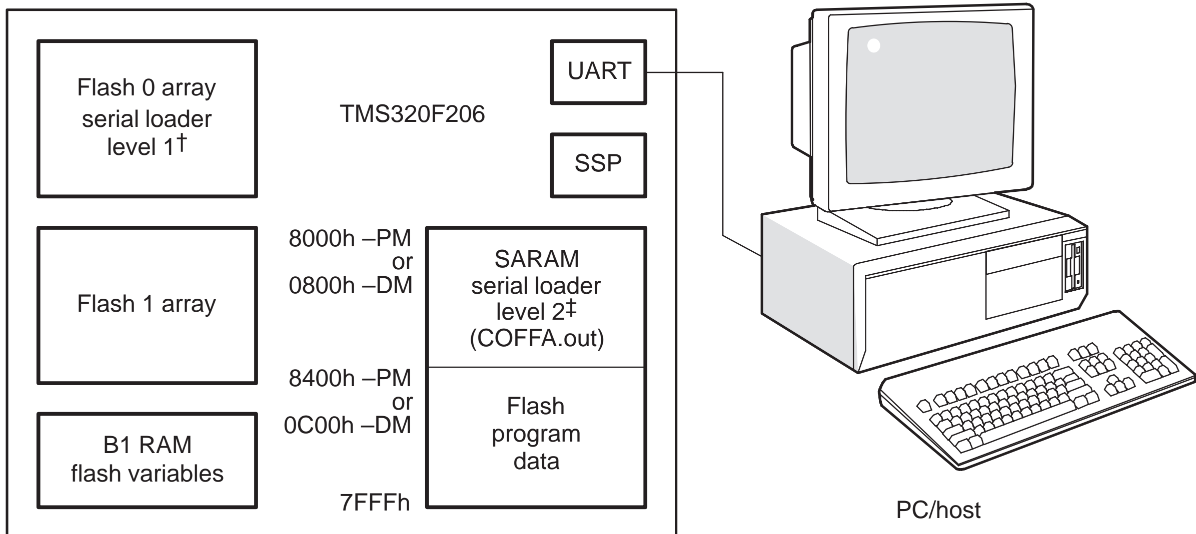
The serial loader for the TMS320F206 device facilitates initial programming of flash arrays. This section describes functional aspects of the serial loader and gives a quick start for flash programming.

B.1.1 Revision 2.0 Software Features

See the Revision 2.0 serial flash programming and PC/host serial communication utilities on the TI web for details and source at **www.ti.com** under *C2000 DSP devices*.

B.1.2 ’F206 Memory Map for the Serial Loader

Figure B–1. ’F206 Memory Map and Serial Port Connections



†Level 1 – Program shipped with Flash 0 array
‡Level 2 – Program that will be loaded using Level 1 code

B.2 Functional Description

B.2.1 Software Modules

The flash serial loader utility is intended for programming the on-chip flash (32k) of the TMS320F206 device. The flash serial loader utility contains three software modules:

- ❑ Serial Loader Level 1
- ❑ Host Serial Communication Utility
- ❑ Serial Loader Level 2

The *serial loader level 1 module* resides in the on-chip flash, specifically flash 0 array at 0x0000h. All the ’F206 devices, rev. 2.0 and above, shipped from TI will contain this code pre–programmed in flash 0. The level 1 module’s serial communication code communicates through the on-chip UART to a host computer to load any application code to its internal memory.

The *host serial communication utility module* (F206sldr.exe) is a Windows ’95 program for IBM/PC compatibles which use PC COM ports to communicate with ’F206 devices. The host utility communicates with level 1 code on the ’F206 device to download flash algorithms and flash data to be programmed.

The *serial loader level 2 module* contains the flash control and flash algorithms. The level 2 code is loaded into internal memory using the level 1 code and host utility.

B.2.2 Operation

Figure B–1 shows a typical configuration between the ’F206 device and a host system. At power on reset, the level 1 software resident in the ’F206’s on-chip flash initializes the UART or the SSP. This initialization is contingent on the status of the BIO pin. If the BIO pin is high, the UART loader is enabled. The UART loader enables auto-baud detect logic and waits for characters through the UART port. Figure B–2 explains the software logic in detail.

The host PC sends ascii character ’a’ as the first character through the serial link to the ’F206. On receipt of a valid ’a’ the level 1 software logic locks to incoming data rate, updates its baud rate register, and echoes character ’a’ back to the host. After receiving a valid echo from the DSP, the host sends the level 2 algorithm code to the ’F206. The level 2 code takes control of the DSP core. The level 2 code handshakes with the host to receive flash data for flash programming.

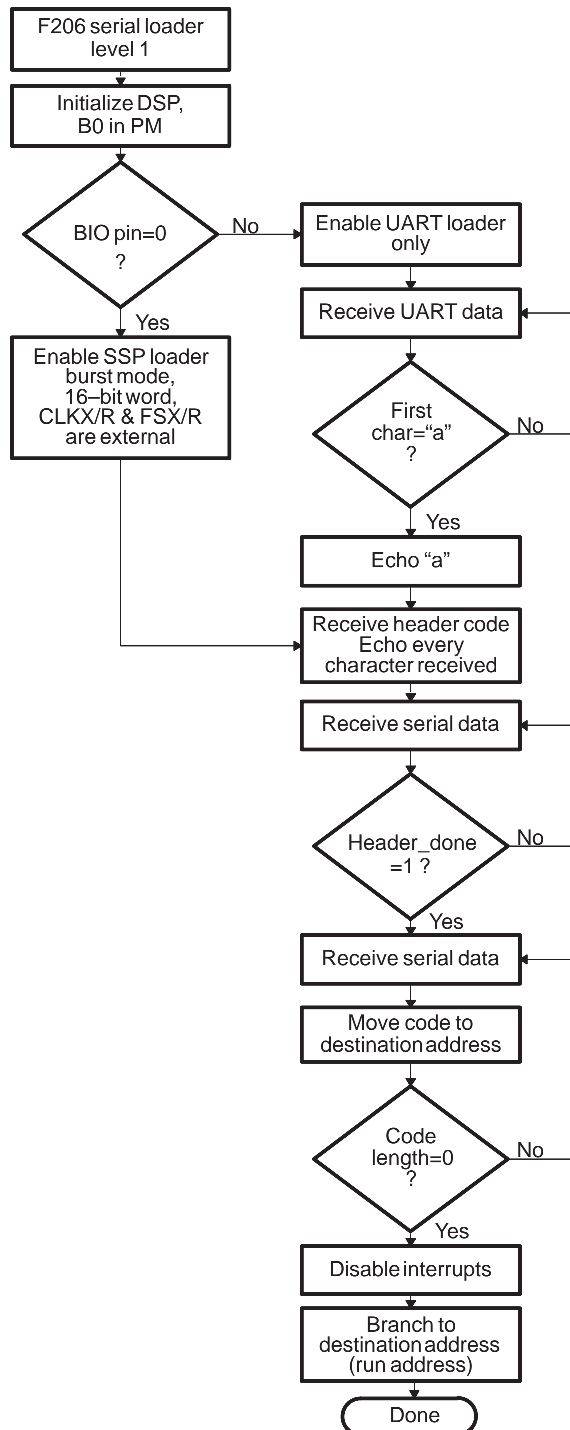
B.2.3 Host Utility Loading Status and Modes

The host utility communicates with level 2 code until the programming is done and updates the communication status window (both successful completion

and error code, if any). The host activates its communication status through the DTR and RTS signals on its serial port as well as on the host monitor. The DTR signal goes low when it receives a valid echo of character ’a’ from the ’F206 device. This indicates that band lock is successful and DTS remains low until the flash programming is complete. If during the loading process any error occurs, the RTS signal goes active low. It remains low indicating that there was an error in the current loading process. If LEDs (light emitting devices) are added to these signals, they provide visual indication of loading status at the remote end (at the ’F206 device side).

The host software runs in either continuous mode or single device mode. In the single device mode, the host program halts after loading/programming the device. In the continuous mode, the host software resends character ’a’ and waits for a valid echo to proceed with the loading process. This logic runs continuously until the program is aborted. The continuous mode enables multiple device programming without manual interaction on the host terminal.

Figure B–2. TMS320F206 Flash Serial Loader – 'F206 Level 1 Flow Chart



B.3 Serial Loader Code

B.3.1 ’F206 Serial Loader Code – Level 1

```
* Program : usload_2.asm                                      *
* Function: F206 Serial loader Code –Level 1                  *
*          Loads code/data either through UART or SSP         *
*          if BIO pin is low at boot/reset time               *
*          Uart starts in autobaud mode, receive              *
*          ”a” or ”A”as first character.                      *
*          The cpu will wait indefinitely for first           *
*          character to be ”a” or”A”. On receipt of ”a” or*
*          ”A” uart data will be loaded as  valid code.       *
*          Receive data format :                              *
*          First character       ”A” or ”a”                   *
*          Header : Destination/Load/Run                      *
*                   start address        1 word               *
*                   Program code/length  2 word               *
*                   Program code/data    from 3 word          *
*          After data load the, interrupts will be            *
*          disabled and PC will jump to the Destination       *
*          /Load/Run address.                                 *
*                                                             *
* Revision  : 1.1                                             *
* Written by: Sam Saba/ASP/St,TX           Date: 7/17/97      *

       .title  ” Serial loader”  ; Title
       .copy ”finit.h”           ; Variable and register declaration
       .text
       b start
       b      inpt1        ; INT1 – These interrupt vector locations
                           ;        are with RET, for safety.
       b      inpt23       ; INT2/INT3 – The exact interrupt routine address
                           ;             need to be specified here when
                           ;             interrupt routines are used
       b      time         ; TINI  Timer interrupt
       b      codrx        ; RX_Sync interrupt
                           ;
       b      codtx        ; TX_SYNC interrupt
       b      uart         ; TX/RX Uart port interrupt
start: setc CNF            ; Block B0 in PM
       ldp #0h             ; set DP=0
       setc INTM           ; Disable all interrupts
* UART initialization *
       splk #0ffffh,ifr    ; clear interrupts
       splk #0000h,B2S_0
       out B2S_0, wsgr     ; Set zero wait states
       splk #0006h,B2S_0
       out B2S_0, pmst     ; Set SARAM in DM and PM
*Uart initialization with autobaud enable
       splk #0c0a0h,B2S_0  ; reset the UART by writing 0
       out B2S_0, aspcr    ; 1 stop bit, rx interrupt, input i/o
       splk #0e0a0h,B2S_0  ; CDC=1 enable
```

```
        out B2S_0,aspcr
        splk #4fffh,B2S_0            ; enable ADC bit
        out B2S_0,iosr               ; enable auto baud
        splk #20h,imr                ; Enable UART interrupt only
        bcnd sspld,bio               ; If BIO is low use SSP loader
        b uartld
*SSP initialization, if BIO pin = 0 at boot/reset, else UART loader enabled
sspld:  splk #0c00ah,B2S_0           ; Initialize SSP in Burst mode
        out B2S_0,sspcr              ; External Clocks, 16 bit word
        splk #0c03ah, B2S_0          ; Interrupt on 1 word in FIFO
        out B2S_0, sspcr
        splk #8h,imr                 ; Enable SSP RX interrupt only
uartld: lacc #0
        lar ar1,#B2                  ; Point B2_RAM start address
        mar *,ar1
        rpt #16
        sacl *+                      ; Clear B2 memory
        lar ar1,#00h                 ; Clear pointers
        lar ar2,#00h                 ;
        lar ar3,#00h
        clrc intm
wait:   idle                         ;
        bit B2FM_8,15                ; Wait until Data_move ready flag
        bcnd wait,ntc
        splk #0,B2FM_8
        lacl B2PA_2                  ; Load destination address
        tblw B2PD_5                  ; Move data to the current destination address
        add #1                       ; Increment destination address+1
        sacl B2PA_2                  ; save next destination address
        banz wait,*-
        setc intm                    ; Disable interrupts
        lacl B2PA_3                  ; Point to Destination/Load/Run address
        bacc                         ; Branch to Program address
        b wait
uart:
        in B2S_0,aspcr
        bit B2S_0,10                 ; Check CDC =1
        bcnd nrcv,ntc                ; If 0 , start receive, autobaud done
        in B2S_1,iosr                ; load input status from iosr
        bit B2S_1,1                  ; check if auto baud bit is set,else return
        bcnd nauto,ntc               ; and wait for Auto baud detect receive
        splk #4000h,B2S_1            ; Auto baud detect done
        out B2S_1,iosr               ; clear ADC
        splk #0e080h,B2S_1
        out B2S_1, aspcr             ; Disable CDC bit/ auto baud
        in B2S_1,adtr                ; Dummy read to discard ”a”
        out B2S_1,adtr               ; Echo back ”a”
nauto:  in B2S_1,adtr                ; Dummy read to clear uart rx buffer
        b skip                       ; Exit and wait for ”a”
nrcv:   in B2S_0,iosr                ; Load input status from iosr
        bit B2S_0,7                  ; bit 8 in the data
        bcnd skip,ntc                ; IF DR=0 no echo, return
        mar *,ar1                    ; Valid UART data, Point to Word index reg.
        bit B2D_6,15                 ; Check if bit0 of word index =1,low byte
```

```
        bcnd lbyte,tc       ; received!
        in B2S_1,adtr       ; No, Hi byte received!
        out B2S_1,adtr      ; Echo receive data
        lacc B2S_1,8        ; Align to upper byte
        sacl B2D_7          ; Save aligned word
        mar *+              ; Increment Word Index
        sar ar1,B2D_6       ; Store high_byte flag
        splk #0,B2FM_8      ; Reset Data/word move flag as only hi-byte recd!
        b skip              ; wait for next byte
lbyte:  in B2S_0,adtr       ; Receive second byte/low byte
        out B2S_0,adtr      ; Echo received data
        lacc B2S_0,0
        and #0ffh           ; Clear high byte
        or B2D_7            ; Add high byte to the word
        sacl B2PD_5         ; store 16-bit word at ar1
        mar *+              ; 1+
        sar ar1,B2D_6       ; Save the count
        bit B2FH_9,15       ; Check Header_done flag
        bcnd smove,tc       ; No, if 2 words received update Data_move flag
        lar ar0,#2
        cmpr 0
        bcnd word2,ntc
        sacl B2PA_2         ; Store destination/Load/Run address
        sacl B2PA_3         ; Store destination/Load/Run address
        b skip              ;
word2:  lar ar0,#4          ; Check if 4 words recvd, update program length
        cmpr 0              ; Program length register
        bcnd skip,ntc       ; Else exit
        lar ar2,B2PD_5      ; Yes received!,Load PM length in AR2
        sar ar2, B2PL_4     ; Save program length
        splk #1,B2FH_9      ; Set Header_done flag
        b skip
smove:  mar *,ar2
        splk #1h,B2FM_8     ; Set UART Data_move ready flag
skip:   splk #6600h,B2S_0
        out B2S_0,iosr      ; Clear all Interrupt sources
        splk #0020h, ifr    ; Clear interrupt in ifr!
        clrc intm
        ret
* SSP loader code!
codrx:
        in B2S_0,sdtr       ; Load Scratch register
        out B2S_0,sdtr      ; Echo received data
        mar *,ar3           ; Set Word index register as ar3
        mar *+              ; Increment word index
        lar ar0,#1          ; If word index =1 save Program start address
        cmpr 0
        bcnd pmad,tc
        lar ar0,#2          ; If index =2 save Program length
        cmpr 0
        bcnd plen,tc
        lacc B2S_0,0
        sacl B2PD_5,0       ; Store received word
        splk #1h,B2FM_8     ; Set SSP Data_move ready flag
```

```
        b skips,ar2
pmad:   lacc B2S_0,0           ; Store destinations start address at
        sacl B2PA_2            ; B2PA_2 and B2PA_3
        sacl B2PA_3
        b skips,ar2
plen:   lar ar2,B2S_0          ; Store Program length at B2PL_4
        sar ar2,B2PL_4
skips:  splk #8h,ifr           ; Clear interrupt flag
        clrc intm
        ret
inpt1:  ret
inpt23: ret
time:   ret
codtx:  ret
        .end                   ; Assembler module end directive –optional
```

B.3.2 ’F206 Serial Loader Code – Level 1 Only

```
* Include file with I/O register declarations *
* For usload_2.asm Serial loader Level 1 only *
*                                             *
* Written by Sam Saba,TI Houston      4/17/97 *
       .mmregs
* On–chip register equates
*Flash control registers
f_access0     .set   0ffe0h
f_access1     .set   0ffe1h
pmst          .set   0ffe4h
* CLKOUT
clk1          .set   0ffe8h
* INTERRUPT CONTROL
icr           .set   0ffech
* SYNC PORT
sdtr          .set   0fff0h
sspcr         .set   0fff1h
* UART
adtr          .set   0fff4h
aspcr         .set   0fff5h
iosr          .set   0fff6h
brd           .set   0fff7h
* TIMER
tcr           .set   0fff8h
prd           .set   0fff9h
tim           .set   0fffah
* WAIT STATES
wsgr          .set   0fffch
* Variables
B2            .set 60h
B2S_0         .set B2+0h             ; Scratch registers
B2S_1         .set B2+1h
B2PA_2        .set B2+2h             ; Program start address
B2PA_3        .set B2+3h             ; Program start address
B2PL_4        .set B2+4h             ; Program Length
B2PD_5        .set B2+5h             ; Program Code/Data
B2D_6         .set B2+6h             ; Variables
B2D_7         .set B2+7h
B2FM_8        .set B2+8h             ; Flag for start Data move – Data_move
B2FH_9        .set B2+9h             ; Flag for Header receive – Header_done
B2FD_a        .set B2+0ah            ; Flag for data move complete – Data_ready
B2FSH         .set B2+0bh            ; High word check sum
B2FSL         .set B2+0ch            ; Low word check sum
```

Appendix C

TMS320C1x/C2x/C20x/C5x Instruction Set Comparison

This appendix contains a table that compares the TMS320C1x, TMS320C2x, TMS320C20x, and TMS320C5x instructions alphabetically. Each table entry shows the syntax for the instruction, indicates which devices support the instruction, and describes the operation of the instruction. Section C.1 shows a sample table entry and describes the symbols and abbreviations used in the table.

The TMS320C2x, TMS320C20x, and TMS320C5x devices have *enhanced instructions*; enhanced instructions are single mnemonics that perform the functions of several similar instructions. Section C.2 summarizes the enhanced instructions.

This appendix does not cover topics such as opcodes, instruction timing, or addressing modes; in addition to this book, the following documents cover such topics in detail:

TMS320C1x User's Guide (literature number SPRU013)

TMS320C2x User's Guide (literature number SPRU014)

TMS320C5x User's Guide (literature number SPRU056)

Topic **Page**

C.1 Using the Instruction Set Comparison Table C-2

C.2 Enhanced Instructions C-5

C.3 Instruction Set Comparison Table C-6

C.1 Using the Instruction Set Comparison Table

To help you read the comparison table, this section provides an example of a table entry and a list of acronyms.

C.1.1 An Example of a Table Entry

In cases where more than one syntax is used, the first syntax is usually for direct addressing and the second is usually for indirect addressing. Where three or more syntaxes are used, the syntaxes are normally specific to a device.

This is how the AND instruction appears in the table:

Syntax	1x	2x	2xx	5x	Description
AND *dma*	√	√	√	√	**AND With Accumulator**
AND {*ind*} [, *next ARP*]	√	√	√	√	TMS320C1x and TMS320C2x devices: AND the contents of the addressed data-memory location with the 16 LSBs of the accumulator. The 16 MSBs of the accumulator are ANDed with 0s.
AND **#***lk* [, *shift*]			√	√	TMS320C20x and TMS320C5x devices: AND the contents of the addressed data-memory location or a 16-bit immediate value with the contents of the accumulator. The 16 MSBs of the accumulator are ANDed with 0s. If a shift is specified, left shift the constant before the AND. Low-order bits below and high-order bits above the shifted value are treated as 0s.

The first column, *Syntax*, states the mnemonic and the syntaxes for the AND instruction.

The checks in the second through the fifth columns, *1x*, *2x*, *2xx,* and *5x*, indicate the devices that can be used with each of the syntaxes.

1x refers to the TMS320C1x devices
2x refers to the TMS320C2x devices, including TMS320C25
2xx refers to the TMS320C20x devices
5x refers to the TMS320C5x devices

In this example, you can use the first two syntaxes with TMS320C1x, TMS320C2x, TMS320C20x, and TMS320C5x devices, but you can use the last syntax only with TMS320C20x and TMS320C5x devices.

The sixth column, *Description*, briefly describes how the instruction functions. Often, an instruction functions slightly differently for the different devices: read the entire description before using the instruction.

C.1.2 Symbols and Acronyms Used in the Table

The following table lists the instruction set symbols and acronyms used throughout this chapter:

Table C–1. Symbols and Acronyms Used in the Instruction Set Comparison Table

Symbol	Description	Symbol	Description
lk	16-bit immediate value	INTM	interrupt mask bit
k	8-bit immediate value	INTR	interrupt mode bit
{ind}	indirect address	OV	overflow bit
ACC	accumulator	P	program bus
ACCB	accumulator buffer	PA	port address
AR	auxiliary register	PC	program counter
ARCR	auxiliary register compare	PM	product shifter mode
ARP	auxiliary register pointer	pma	program-memory address
BMAR	block move address register	RPTC	repeat counter
BRCR	block repeat count register	shift, $shift_n$	shift value
C	carry bit	src	source address
DBMR	dynamic bit manipulation register	ST	status register
dma	data-memory address	SXM	sign-extension mode bit
DP	data-memory page pointer	TC	test/control bit
dst	destination address	T	temporary register
FO	format status list	TREGn	TMS320C5x temporary register (0–2)
FSX	external framing pulse	TXM	transmit mode status register
IMR	interrupt mask register	XF	XF pin status bit

Based on the device, this is how the indirect addressing operand {ind} is interpreted:

{ind}	**’C1x**:	{ * | *+ | *– }
	’C2x:	{ * | *+ | *– | *0+| *0– | *BR0+ | *BR0– }
	’C20x:	{ * | *+ | *– | *0+| *0– | *BR0+ | *BR0– }
	’C5x:	{ * | *+ | *– | *0+| *0– | *BR0+ | *BR0– }

where the possible options are separated by vertical bars (|). For example:

`ADD {ind}`

is interpreted as:

’C1x devices	ADD { * | *+ | *– }
’C2x devices	ADD { * | *+ | *– | *0+ | *0– | *BR0+ | *BR0– }
’C20x devices	ADD { * | *+ | *– | *0+ | *0– | *BR0+ | *BR0– }
’C5x devices	ADD { * | *+ | *– | *0+ | *0– | *BR0+ | *BR0– }

Based on the device, these are the sets of values for shift, $shift_1$, and $shift_2$:

shift	**’C1x**:	0–15 (shift of 0–15 bits)
	’C2x:	0–15 (shift of 0–15 bits)
	’C20x:	0–16 (shift of 0–16 bits)
	’C5x:	0–16 (shift of 0–16 bits)
$shift_1$	**’C1x**:	n/a
	’C2x:	0–15 (shift of 0–15 bits)
	’C20x:	0–16 (shift of 0–16 bits)
	’C5x:	0–16 (shift of 0–16 bits)
$shift_2$	**’C1x**:	n/a
	’C2x:	n/a
	’C20x:	0–15 (shift of 0–15 bits)
	’C5x:	0–15 (shift of 0–15 bits)

In some cases, the sets are smaller; in these cases, the valid sets are given in the *Description* column of the table.

C.2 Enhanced Instructions

An enhanced instruction is a single mnemonic that performs the functions of several similar instructions. For example, the enhanced instruction ADD performs the ADD, ADDH, ADDK, and ADLK functions and replaces any of these other instructions at assembly time. For example, when a program using ADDH is assembled for the ’C20x or ’C5x, ADDH is replaced by an ADD instruction that performs the same function. These enhanced instructions are valid for TMS320C2x, TMS320C20x, and TMS320C5x devices (not TMS320C1x).

Table C–2 below summarizes the enhanced instructions and the functions that the enhanced instructions perform (based on TMS320C1x/2x mnemonics).

Table C–2. Summary of Enhanced Instructions

Enhanced Instruction	Includes These Operations
ADD	ADD, ADDH, ADDK, ADLK
AND	AND, ANDK
BCND	BBNZ, BBZ, BC, BCND, BGEZ, BGZ, BIOZ, BLEZ, BLZ, BNC, BNV, BNZ, BV, BZ
BLDD	BLDD, BLKD
BLDP	BLDP, BLKP
CLRC	CLRC, CNFD, EINT, RC, RHM, ROVM, RSXM, RTC, RXF
LACC	LAC, LACC, LALK, ZALH
LACL	LACK, LACL, ZAC, ZALS
LAR	LAR, LARK, LRLK
LDP	LDP, LDPK
LST	LST, LST1
MAR	LARP, MAR
MPY	MPY, MPYK
OR	OR, ORK
RPT	RPT, RPTK
SETC	CNFP, DINT, SC, SETC, SHM, SOVM, SSXM, STC, SXF
SUB	SUB, SUBH, SUBK

C.3 Instruction Set Comparison Table

Syntax	1x	2x	2xx	5x	Description
ABS	√	√	√	√	**Absolute Value of Accumulator** If the contents of the accumulator are less than zero, replace the contents with the 2s complement of the contents. If the contents are ≥ 0, the accumulator is not affected.
ADCB				√	**Add ACCB to Accumulator With Carry** Add the contents of the ACCB and the value of the carry bit to the accumulator. If the result of the addition generates a carry from the accumulator's MSB, the carry bit is set to 1.
ADD *dma* [, *shift*]	√	√	√	√	**Add to Accumulator With Shift**
ADD {*ind*} [, *shift* [, *next ARP*]]	√	√	√	√	TMS320C1x and TMS320C2x devices: Add the contents of the addressed data-memory location to the accumulator; if a shift is specified, left shift the contents of the location before the add. During shifting, low-order bits are zero filled, and high-order bits are sign extended.
ADD # *k*			√	√	TMS320C20x and TMS320C5x devices: Add the contents of the addressed data-memory location or an immediate value to the accumulator; if a shift is specified, left shift the data before the add. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
ADD # *lk* [, *shift2*]			√	√	
ADDB				√	**Add ACCB to Accumulator** Add the contents of the ACCB to the accumulator.
ADDC *dma*		√	√	√	**Add to Accumulator With Carry**
ADDC {*ind*} [, *next ARP*]		√	√	√	Add the contents of the addressed data-memory location and the carry bit to the accumulator.
ADDH *dma*	√	√	√	√	**Add High to Accumulator**
ADDH {*ind*} [, *next ARP*]	√	√	√	√	Add the contents of the addressed data-memory location to the 16 MSBs of the accumulator. The LSBs are not affected. If the result of the addition generates a carry, the carry bit is set to 1. TMS320C2x, TMS320C20x, and TMS320C5x devices: If the result of the addition generates a carry from the accumulator's MSB, the carry bit is set to 1.

Syntax	1x	2x	2xx	5x	Description
ADDK #*k*		√	√	√	**Add to Accumulator Short Immediate** TMS320C1x devices: Add an 8-bit immediate value to the accumulator. TMS320C2x, TMS320C20x, and TMS320C5x devices: Add an 8-bit immediate value, right justified, to the accumulator with the result replacing the accumulator contents. The immediate value is treated as an 8-bit positive number; sign extension is suppressed.
ADDS *dma* **ADDS** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Add to Accumulator With Sign Extension Suppressed** Add the contents of the addressed data-memory location to the accumulator. The value is treated as a 16-bit unsigned number; sign extension is suppressed.
ADDT *dma* **ADDT** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Add to Accumulator With Shift Specified by T Register** Left shift the contents of the addressed data-memory location by the value in the 4 LSBs of the T register; add the result to the accumulator. If a shift is specified, left shift the data before the add. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1. TMS320C20x and TMS320C5x devices: If the result of the addition generates a carry from the accumulator's MSB, the carry bit is set to 1.
ADLK #*lk* [, *shift*]		√	√	√	**Add to Accumulator Long Immediate With Shift** Add a 16-bit immediate value to the accumulator; if a shift is specified, left shift the value before the add. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
ADRK #*k*		√	√	√	**Add to Auxiliary Register Short Immediate** Add an 8-bit immediate value to the current auxiliary register.

Syntax	1x	2x	2xx	5x	Description
AND *dma* **AND** {*ind*} [, *next ARP*] **AND** **#***lk* [, *shift*]	√ √ 	√ √ 	√ √ √	√ √ √	**AND With Accumulator** TMS320C1x and TMS320C2x devices: AND the contents of the addressed data-memory location with the 16 LSBs of the accumulator. The 16 MSBs of the accumulator are ANDed with 0s. TMS320C20x and TMS320C5x devices: AND the contents of the addressed data-memory location or a 16-bit immediate value with the contents of the accumulator. The 16 MSBs of the accumulator are ANDed with 0s. If a shift is specified, left shift the constant before the AND. Low-order bits below and high-order bits above the shifted value are treated as 0s.
ANDB				√	**AND ACCB to Accumulator** AND the contents of the ACCB to the accumulator.
ANDK **#***lk* [, *shift*]		√	√	√	**AND Immediate With Accumulator With Shift** AND a 16-bit immediate value with the contents of the accumulator; if a shift is specified, left shift the constant before the AND.
APAC	√	√	√	√	**Add P Register to Accumulator** Add the contents of the P register to the accumulator. TMS320C2x, TMS320C20x, and TMS320C5x devices: Before the add, left shift the contents of the P register as defined by the PM status bits.
APL [**#***lk*] ,*dma* **APL** [**#***lk,*] {*ind*} [, *next ARP*]				√ √	**AND Data-Memory Value With DBMR or Long Constant** AND the data-memory value with the contents of the DBMR or a long constant. If a long constant is specified, it is ANDed with the contents of the data-memory location. The result is written back into the data-memory location previously holding the first operand. If the result is 0, the TC bit is set to 1; otherwise, the TC bit is cleared.
B *pma* **B** *pma* [, {*ind*} [, *next ARP*]]	√ 	 √	 √		**Branch Unconditionally** Branch to the specified program-memory address. TMS320C2x and TMS320C20x devices: Modify the current AR and ARP as specified.

Syntax	1x	2x	2xx	5x	Description
B[*D*] *pma* [, {*ind*} [, *next ARP*]]				√	**Branch Unconditionally With Optional Delay** Modify the current auxiliary register and ARP as specified and pass control to the designated program-memory address. If you specify a delayed branch (BD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before branching.
BACC		√	√		**Branch to Address Specified by Accumulator** Branch to the location specified by the 16 LSBs of the accumulator.
BACC[*D*]				√	**Branch to Address Specified by Accumulator With Optional Delay** Branch to the location specified by the 16 LSBs of the accumulator. If you specify a delayed branch (BACCD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before branching.
BANZ *pma*	√				**Branch on Auxiliary Register Not Zero** If the contents of the 9 LSBs of the current auxiliary register (TMS320C1x) or the contents of the entire current auxiliary register (TMS320C2x) are ≠ 0, branch to the specified program-memory address. TMS320C2x and TMS320C20x devices: Modify the current AR and ARP (if specified) or decrement the current AR (default). TMS320C1x devices: Decrement the current AR.
BANZ *pma* [, {*ind*} [, *next ARP*]]		√	√		
BANZ[*D*] *pma* [, {*ind*} [, *next ARP*]]				√	**Branch on Auxiliary Register Not Zero With Optional Delay** If the contents of the current auxiliary register are ≠ 0, branch to the specified program-memory address. Modify the current AR and ARP as specified, or decrement the current AR. If you specify a delayed branch (BANZD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before branching.

Syntax	1x	2x	2xx	5x	Description
BBNZ *pma* [, {*ind*} [, *next ARP*]]		√	√	√	**Branch on Bit ≠ Zero** If the TC bit = 1, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: If the –p porting switch is used, modify the current AR and ARP as specified.
BBZ *pma* [, {*ind*} [, *next ARP*]]		√	√	√	**Branch on Bit = Zero** If the TC bit = 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BBZ *pma*				√	
BC *pma* [, {*ind*} [, *next ARP*]]		√		√	**Branch on Carry** If the C bit = 1, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BC *pma*			√	√	
BCND *pma*, $cond_1$ [, $cond_2$] [, ...]			√		**Branch Conditionally** Branch to the program-memory address if the specified conditions are met. Not all combinations of conditions are meaningful.
BCND[*D*] *pma*, $cond_1$ [, $cond_2$] [, ...]				√	**Branch Conditionally With Optional Delay** Branch to the program-memory address if the specified conditions are met. Not all combinations of conditions are meaningful. If you specify a delayed branch (BCNDD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before branching.

Syntax	1x	2x	2xx	5x	Description
BGEZ *pma*	√		√	√	**Branch if Accumulator ≥ Zero**
BGEZ *pma* [, {*ind*} [, *next ARP*]]		√		√	If the contents of the accumulator ≥ 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BGZ *pma*	√		√	√	**Branch if Accumulator > Zero**
BGZ *pma* [, {*ind*} [, *next ARP*]]		√		√	If the contents of the accumulator are > 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BIOZ *pma*	√		√	√	**Branch on I/O Status = Zero**
BIOZ *pma* [, {*ind*} [, *next ARP*]]		√		√	If the $\overline{\text{BIO}}$ pin is low, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BIT *dma, bit code*		√	√	√	**Test Bit**
BIT {*ind*}, *bit code* [, *next ARP*]		√	√	√	Copy the specified bit of the data-memory value to the TC bit in ST1.
BITT *dma*		√	√	√	**Test Bit Specified by T Register**
BITT {*ind*} [, *next ARP*]		√	√	√	TMS320C2x and TMS320C20x devices: Copy the specified bit of the data-memory value to the TC bit in ST1. The 4 LSBs of the T register specify which bit is copied. TMS320C5x devices: Copy the specified bit of the data-memory value to the TC bit in ST1. The 4 LSBs of the TREG2 specify which bit is copied.

Syntax	1x	2x	2xx	5x	Description
BLDD **#***lk, dma* **BLDD** **#***lk*, {*ind*} [*, next ARP*] **BLDD** *dma,* **#***lk* **BLDD** {*ind*}*,* **#***lk* [*, next ARP*] **BLDD** BMAR, *dma* **BLDD** BMAR, {*ind*} [*, next ARP*] **BLDD** *dma* BMAR **BLDD** {*ind*}*,* BMAR [*, next ARP*]			√ √ √ √	√ √ √ √ √ √ √ √	**Block Move From Data Memory to Data Memory** Copy a block of data memory into data memory. The block of data memory is pointed to by *src*, and the destination block of data memory is pointed to by *dst*. TMS320C20x devices: The word of the source and/or the destination space can be pointed to with a long immediate value or a data-memory address. You can use the RPT instruction with BLDD to move consecutive words, pointed to indirectly in data memory, to a contiguous program-memory space. The number of words to be moved is 1 greater than the number contained in the RPTC at the beginning of the instruction. TMS320C5x devices: The word of the source and/or the destination space can be pointed to with a long immediate value, the contents of the BMAR, or a data-memory address. You can use the RPT instruction with BLDD to move consecutive words, pointed to indirectly in data memory, to a contiguous program-memory space. The number of words to be moved is 1 greater than the number contained in the RPTC at the beginning of the instruction.
BLDP *dma* **BLDP** {*ind*} [*, next ARP*]				√ √	**Block Move From Data Memory to Program Memory** Copy a block of data memory into program memory pointed to by the BMAR. You can use the RPT instruction with BLDP to move consecutive words, indirectly pointed to in data memory, to a contiguous program-memory space pointed to by the BMAR.
BLEZ *pma* **BLEZ** *pma* [*,* {*ind*} [*, next ARP*]]	√ 	 √	√ √	√ √	**Branch if Accumulator ≤ Zero** If the contents of the accumulator are ≤ 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.

Syntax	1x	2x	2xx	5x	Description
BLKD *dma1, dma2* **BLKD** *dma1,* {*ind*} [*, next ARP*]	 	√ √	√ √	√ √	**Block Move From Data Memory to Data Memory** Move a block of words from one location in data memory to another location in data memory. Modify the current AR and ARP as specified. RPT or RPTK must be used with BLKD, in the indirect addressing mode, if more than one word is to be moved. The number of words to be moved is 1 greater than the number contained in RPTC at the beginning of the instruction.
BLKP *pma, dma* **BLKP** *pma,* {*ind*} [*, next ARP*]	 	√ √	√ √	√ √	**Block Move From Program Memory to Data Memory** Move a block of words from a location in program memory to a location in data memory. Modify the current AR and ARP as specified. RPT or RPTK must be used with BLKD, in the indirect addressing mode, if more than one word is to be moved. The number of words to be moved is 1 greater than the number contained in RPTC at the beginning of the instruction.
BLPD† **#***pma, dma* **BLPD**† **#***pma,* {*ind*} [*, next ARP*] **BLPD**† BMAR, *dma* **BLPD**† BMAR, {*ind*} [*, next ARP*]	 	 	√ √ 	√ √ √ √	**Block Move From Program Memory to Data Memory** Copy a block of program memory into data memory. The block of program memory is pointed to by *src*, and the destination block of data memory is pointed to by *dst*. TMS320C20x devices: The word of the source space can be pointed to with a long immediate value. You can use the RPT instruction with BLPD to move consecutive words that are pointed at indirectly in data memory to a contiguous program-memory space. TMS320C5x devices: The word of the source space can be pointed to with a long immediate value or the contents of the BMAR. You can use the RPT instruction with BLPD to move consecutive words that are pointed at indirectly in data memory to a contiguous program-memory space.
BLZ *pma* **BLZ** *pma* [*,* {*ind*} [*, next ARP*]]	√ 	 √	√ √	√ 	**Branch if Accumulator < Zero** If the contents of the accumulator are < 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.

† BLDD and BLPD are TMS320C5x and TMS320C20x instructions for the BLKD and BLKP instructions in the TMS320C2x and TMS320C1 devices. The assembler converts TMS320C2x code to BLKB and BLKP.

Syntax	1x	2x	2xx	5x	Description
BNC *pma* [, {*ind*} [, *next ARP*]]		√	√	√	**Branch on No Carry** If the C bit = 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BNV *pma* [, {*ind*} [, *next ARP*]]		√	√	√	**Branch if No Overflow** If the OV flag is clear, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BNZ *pma*	√				**Branch if Accumulator ≠ Zero**
BNZ *pma* [, {*ind*} [, *next ARP*]]		√	√	√	If the contents of the accumulator ≠ 0, branch to the specified program-memory address. TMS320C2x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified when the –p porting switch is used.
BSAR [*shift*]				√	**Barrel Shift** In a single cycle, execute a 1- to 16-bit right arithmetic barrel shift of the accumulator. The sign extension is determined by the sign-extension mode bit in ST1.
BV *pma*	√				**Branch on Overflow**
BV *pma* [, {*ind*} [, *next ARP*]]		√	√	√	If the OV flag is set, branch to the specified program-memory address and clear the OV flag. TMS320C2x, TMS320C20x, and TMS320C5x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: To modify the AR and ARP, use the –p porting switch.

Syntax	1x	2x	2xx	5x	Description
BZ *pma*	√		√	√	**Branch if Accumulator = Zero**
BZ *pma* [, {*ind*} [, *next ARP*]]		√			If the contents of the accumulator = 0, branch to the specified program-memory address. TMS320C2x, TMS320C20x and TMS320C5x devices: Modify the current AR and ARP as specified. TMS320C20x and TMS320C5x devices: To modify the AR and ARP, use the –p porting switch.
CALA	√	√	√		**Call Subroutine Indirect** The contents of the accumulator specify the address of a subroutine. Increment the PC, push the PC onto the stack, then load the 12 (TMS320C1x) or 16 (TMS320C2x/C20x) LSBs of the accumulator into the PC.
CALA[*D*]				√	**Call Subroutine Indirect With Optional Delay** The contents of the accumulator specify the address of a subroutine. Increment the PC and push it onto the stack; then load the 16 LSBs of the accumulator into the PC. If you specify a delayed branch (CALAD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before the call.
CALL *pma*	√				**Call Subroutine**
CALL *pma* [,{*ind*} [, *next ARP*]]		√	√		The contents of the addressed program-memory location specify the address of a subroutine. Increment the PC by 2, push the PC onto the stack, then load the specified program-memory address into the PC. TMS320C2x and TMS320C20x devices: Modify the current AR and ARP as specified.
CALL[*D*] *pma* [, {*ind*} [, *next ARP*]]				√	**Call Unconditionally With Optional Delay** The contents of the addressed program-memory location specify the address of a subroutine. Increment the PC and push the PC onto the stack; then load the specified program-memory address (symbolic or numeric) into the PC. Modify the current AR and ARP as specified. If you specify a delayed branch (CALLD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before the call.

Syntax	1x	2x	2xx	5x	Description
CC *pma, $cond_1$* [, *$cond_2$*] [, ...]			√		**Call Conditionally** If the specified conditions are met, control is passed to the pma. Not all combinations of conditions are meaningful.
CC[*D*] *pma*, *$cond_1$* [, *$cond_2$*] [, ...]				√	**Call Conditionally With Optional Delay** If the specified conditions are met, control is passed to the pma. Not all combinations of conditions are meaningful. If you specify a delayed branch (CCD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before the call.
CLRC *control bit*			√	√	**Clear Control Bit** Set the specified control bit to a logic 0. Maskable interrupts are enabled immediately after the CLRC instruction executes.
CMPL		√	√	√	**Complement Accumulator** Complement the contents of the accumulator (1s complement).
CMPR *CM*		√	√	√	**Compare Auxiliary Register With AR0** Compare the contents of the current auxiliary register to AR0, based on the following cases: If CM = 00_2, test whether AR(ARP) = AR0. If CM = 01_2, test whether AR(ARP) < AR0. If CM = 10_2, test whether AR(ARP) > AR0. If CM = 11_2, test whether AR(ARP) ≠ AR0. If the result is true, load a 1 into the TC status bit; otherwise, load a 0 into the TC bit. The comparison does not affect the tested registers. TMS320C5x devices: Compare the contents of the auxiliary register with the ARCR.
CNFD		√	√	√	**Configure Block as Data Memory** Configure on-chip RAM block B0 as data memory. Block B0 is mapped into data-memory locations 512h–767h. TMS320C5x devices: Block B0 is mapped into data-memory locations 512h–1023h.

Syntax	1x	2x	2xx	5x	Description
CNFP		√	√	√	**Configure Block as Program Memory** Configure on-chip RAM block B0 as program memory. Block B0 is mapped into program-memory locations 65280h–65535h. TMS320C5x devices: Block B0 is mapped into data-memory locations 65024h–65535h.
CONF *2-bit constant*		√			**Configure Block as Program Memory** Configure on-chip RAM block B0/B1/B2/B3 as program memory. For information on the memory mapping of B0/B1/B2/B3, see the *TMS320C2x User's Guide*.
CPL [**#***lk*,] *dma*				√	**Compare DBMR or Immediate With Data Value**
CPL [**#***lk*,] {*ind*} [, *next ARP*]				√	Compare two quantities: If the two quantities are equal, set the TC bit to 1; otherwise, clear the TC bit.
CRGT				√	**Test for ACC > ACCB** Compare the contents of the ACC with the contents of the ACCB, then load the larger signed value into both registers and modify the carry bit according to the comparison result. If the contents of ACC are greater than or equal to the contents of ACCB, set the carry bit to 1.
CRLT				√	**Test for ACC < ACCB** Compare the contents of the ACC with the contents of the ACCB, then load the smaller signed value into both registers and modify the carry bit according to the comparison result. If the contents of ACC are less than the contents of ACCB, clear the carry bit.
DINT	√	√	√	√	**Disable Interrupts** Disable all interrupts; set the INTM to 1. Maskable interrupts are disabled immediately after the DINT instruction executes. DINT does not disable the unmaskable interrupt $\overline{RS}$; DINT does not affect the IMR.
DMOV *dma*	√	√	√	√	**Data Move in Data Memory**
DMOV {*ind*} [, *next ARP*]	√	√	√	√	Copy the contents of the addressed data-memory location into the next higher address. DMOV moves data only within on-chip RAM blocks. TMS320C2x, TMS320C20x, and TMS320C5x devices: The on-chip RAM blocks are B0 (when configured as data memory), B1, and B2.

Syntax	1x	2x	2xx	5x	Description
EINT	√	√	√	√	**Enable Interrupts** Enable all interrupts; clear the INTM to 0. Maskable interrupts are enabled immediately after the EINT instruction executes.
EXAR				√	**Exchange ACCB With ACC** Exchange the contents of the ACC with the contents of the ACCB.
FORT *1-bit constant*		√			**Format Serial Port Registers** Load the FO with a 0 or a 1. If FO = 0, the registers are configured to receive/transmit 16-bit words. If FO = 1, the registers are configured to receive/transmit 8-bit bytes.
IDLE		√	√	√	**Idle Until Interrupt** Forces an executing program to halt execution and wait until it receives a reset or an interrupt. The device remains in an idle state until it is interrupted.
IDLE2				√	**Idle Until Interrupt—Low-Power Mode** Removes the functional clock input from the internal device; this allows for an extremely low-power mode. The IDLE2 instruction forces an executing program to halt execution and wait until it receives a reset or unmasked interrupt.
IN *dma, PA* **IN** {*ind*}, *PA* [, *next ARP*]	√ √	√ √	√ √	√ √	**Input Data From Port** Read a 16-bit value from one of the external I/O ports into the addressed data-memory location. TMS320C1x devices: This is a 2-cycle instruction. During the first cycle, the port address is sent to address lines A2/PA2–A0/PA0; $\overline{\text{DEN}}$ goes low, strobing in the data that the addressed peripheral places on data bus D15–D0. TMS320C2x devices: 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 read. TMS320C20x and TMS320C5x devices: 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.

Syntax	1x	2x	2xx	5x	Description
INTR *K*			√	√	**Soft Interrupt** Transfer program control to the program-memory address specified by *K* (an integer from 0 to 31). This instruction allows you to use your software to execute any interrupt service routine. The interrupt vector locations are spaced apart by two addresses (0h, 2h, 4h, ... , 3Eh), allowing a two-word branch instruction to be placed at each location.
LAC *dma* [, *shift*] **LAC** {*ind*} [, *shift* [, *next ARP*]]	√ √	√ √	√ √	√ √	**Load Accumulator With Shift** Load the contents of the addressed data-memory location into the accumulator. If a shift is specified, left shift the value before loading it into the accumulator. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
LACB				√	**Load Accumulator With ACCB** Load the contents of the accumulator buffer into the accumulator.
LACC *dma* [, $shift_1$] **LACC** {*ind*} [, $shift_1$ [, *next ARP*]] **LACC** **#***lk* [, $shift_2$]		√ √ √	√ √ √	√ √ √	**Load Accumulator With Shift** Load the contents of the addressed data-memory location or the 16-bit constant into the accumulator. If a shift is specified, left shift the value before loading it into the accumulator. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
LACK *8-bit constant*	√	√	√	√	**Load Accumulator Immediate Short** Load an 8-bit constant into the accumulator. The 24 MSBs of the accumulator are zeroed.
LACL *dma* **LACL** {*ind*} [, *next ARP*] **LACL** **#***k*			√ √ √	√ √ √	**Load Low Accumulator and Clear High Accumulator** Load the contents of the addressed data-memory location or zero-extended 8-bit constant into the 16 LSBs of the accumulator. The MSBs of the accumulator are zeroed. The data is treated as a 16-bit unsigned number. TMS320C20x: A constant of 0 clears the contents of the accumulator to 0 with no sign extension.

Syntax	1x	2x	2xx	5x	Description
LACT *dma* **LACT** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Load Accumulator With Shift Specified by T Register** Left shift the contents of the addressed data-memory location by the value specified in the 4 LSBs of the T register; load the result into the accumulator. If a shift is specified, left shift the value before loading it into the accumulator. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
LALK **#***lk* [, *shift*]		√	√	√	**Load Accumulator Long Immediate With Shift** Load a 16-bit immediate value into the accumulator. If a shift is specified, left shift the constant before loading it into the accumulator. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
LAMM *dma* **LAMM** {*ind*} [, *next ARP*]				√ √	**Load Accumulator With Memory-Mapped Register** Load the contents of the addressed memory-mapped register into the low word of the accumulator. The 9 MSBs of the data-memory address are cleared, regardless of the current value of DP or the 9 MSBs of AR (ARP).
LAR *AR, dma* **LAR** *AR,* {*ind*} [, *next ARP*] **LAR** *AR,* **#***k* **LAR** *AR,* **#***lk*	√ √	√ √	√ √ √ √	√ √ √ √	**Load Auxiliary Register** TMS320C1x and TMS320C2x devices: Load the contents of the addressed data-memory location into the designated auxiliary register. TMS320C25, TMS320C20x, and TMS320C5x devices: Load the contents of the addressed data-memory location or an 8-bit or 16-bit immediate value into the designated auxiliary register.
LARK *AR, 8-bit constant*	√	√	√	√	**Load Auxiliary Register Immediate Short** Load an 8-bit positive constant into the designated auxiliary register.
LARP *1-bit constant* **LARP** *3-bit constant*	√	 √	 √	 √	**Load Auxiliary Register Pointer** TMS320C1x devices: Load a 1-bit constant into the auxiliary register pointer (specifying AR0 or AR1). TMS320C2x, TMS320C20x, and TMS320C5x devices: Load a 3-bit constant into the auxiliary register pointer (specifying AR0–AR7).

Syntax	1x	2x	2xx	5x	Description
LDP *dma* **LDP** {*ind*} [, *next ARP*] **LDP** **#***k*	√ √ 	√ √ 	√ √ √	√ √ √	**Load Data-Memory Page Pointer** TMS320C1x devices: Load the LSB of the contents of the addressed data-memory location into the DP register. All high-order bits are ignored. DP = 0 defines page 0 (words 0–127), and DP = 1 defines page 1 (words 128–143/255). TMS320C2x, TMS320C20x, and TMS320C5x devices: Load the 9 LSBs of the addressed data-memory location or a 9-bit immediate value into the DP register. The DP and 7-bit data-memory address are concatenated to form 16-bit data-memory addresses.
LDPK *1-bit constant* **LDPK** *9-bit constant*	√ 	 √	 √	 √	**Load Data-Memory Page Pointer Immediate** TMS320C1x devices: Load a 1-bit immediate value into the DP register. DP = 0 defines page 0 (words 0–127), and DP = 1 defines page 1 (words 128–143/255). TMS320C2x, TMS320C20x, and TMS320C5x devices: Load a 9-bit immediate into the DP register. The DP and 7-bit data-memory address are concatenated to form 16-bit data-memory addresses. DP ≥ 8 specifies external data memory. DP = 4 through 7 specifies on-chip RAM blocks B0 or B1. Block B2 is located in the upper 32 words of page 0.
LMMR *dma,* **#***lk* **LMMR** {*ind*}*,* **#***lk* [*, next ARP*]				√ √	**Load Memory-Mapped Register** Load the contents of the memory-mapped register pointed at by the 7 LSBs of the direct or indirect data-memory value into the long immediate addressed data-memory location. The 9 MSBs of the data-memory address are cleared, regardless of the current value of DP or the 9 MSBs of AR (ARP).
LPH *dma* **LPH** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Load High P Register** Load the contents of the addressed data-memory location into the 16 MSBs of the P register; the LSBs are not affected.
LRLK *AR, lk*		√	√	√	**Load Auxiliary Register Long Immediate** Load a 16-bit immediate value into the designated auxiliary register.
LST *dma* **LST** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Load Status Register** Load the contents of the addressed data-memory location into the ST (TMS320C1x) or into ST0 (TMS320C2x/2xx/5x).

Syntax	1x	2x	2xx	5x	Description
LST **#***n, dma* **LST** **#***n*, {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Load Status Register n** Load the contents of the addressed data-memory location into ST*n*.
LST1 *dma* **LST1** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Load ST1** Load the contents of the addressed data-memory location into ST1.
LT *dma* **LT** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Load T Register** Load the contents of the addressed data-memory location into the T register (TMS320C1x/2x/2xx) or TREG0 (TMS320C5x).
LTA *dma* **LTA** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Load T Register and Accumulate Previous Product** Load the contents of the addressed data-memory location into T register (TMS320C1x/2x/2xx) or TREG0 (TMS320C5x) and add the contents of the P register to the accumulator. TMS320C2x, TMS320C20x, and TMS320C5x devices: Before the add, shift the contents of the P register as specified by the PM status bits.
LTD *dma* **LTD** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Load T Register, Accumulate Previous Product, and Move Data** Load the contents of the addressed data-memory location into the T register (TMS320C1x/2x/2xx) or TREG0 (TMS320C5x), add the contents of the P register to the accumulator, and copy the contents of the specified location into the next higher address (both data-memory locations must reside in on-chip data RAM). TMS320C2x, TMS320C20x, and TMS320C5x devices: Before the add, shift the contents of the P register as specified by the PM status bits.
LTP *dma* **LTP** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Load T Register, Store P Register in Accumulator** Load the contents of the addressed data-memory location into the T register (TMS320C1x/2x/2xx) or TREG0 (TMS320C5x). Store the contents of the product register into the accumulator.
LTS *dma* **LTS** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Load T Register, Subtract Previous Product** Load the contents of the addressed data-memory location into the T register (TMS320C1x/2x/2xx) or TREG0 (TMS320C5x). Shift the contents of the product register as specified by the PM status bits, and subtract the result from the accumulator.

Syntax	1x	2x	2xx	5x	Description
MAC *pma, dma* **MAC** *pma*, {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Multiply and Accumulate** Multiply a data-memory value by a program-memory value and add the previous product (shifted as specified by the PM status bits) to the accumulator.
MACD *dma, pma* **MACD** *pma*, {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Multiply and Accumulate With Data Move** Multiply a data-memory value by a program-memory value and add the previous product (shifted as specified by the PM status bits) to the accumulator. If the data-memory address is in on-chip RAM block B0, B1, or B2, copy the contents of the address to the next higher address.
MADD *dma* **MADD** {*ind*} [, *next ARP*]				√ √	**Multiply and Accumulate With Data Move and Dynamic Addressing** Multiply a data-memory value by a program-memory value and add the previous product (shifted as defined by the PM status bits) into the accumulator. The program-memory address is contained in the BMAR; this allows for dynamic addressing of coefficient tables. MADD functions the same as MADS, with the addition of data move for on-chip RAM blocks.
MADS *dma* **MADS** {*ind*} [, *next ARP*]				√ √	**Multiply and Accumulate With Dynamic Addressing** Multiply a data-memory value by a program-memory value and add the previous product (shifted as defined by the PM status bits) into the accumulator. The program-memory address is contained in the BMAR; this allows for dynamic addressing of coefficient tables.
MAR *dma* **MAR** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Modify Auxiliary Register** Modify the current AR or ARP as specified. MAR acts as NOP in indirect addressing mode.
MPY *dma* **MPY** {*ind*} [, *next ARP*] **MPY #***k* **MPY #***lk*	√ √	√ √	√ √ √ √	√ √ √ √	**Multiply** TMS320C1x and TMS320C2x devices: Multiply the contents of the T register by the contents of the addressed data-memory location; place the result in the P register. TMS320C20x and TMS320C5x devices: Multiply the contents of the T register (TMS320C20x) or TREG0 (TMS320C5x) by the contents of the addressed data-memory location or a 13-bit or 16-bit immediate value; place the result in the P register.

Syntax	1x	2x	2xx	5x	Description
MPYA *dma* **MPYA** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Multiply and Accumulate Previous Product** Multiply the contents of the T register (TMS320C2x/2xx) or TREG0 (TMS320C5x) by the contents of the addressed data-memory location; place the result in the P register. Add the previous product (shifted as specified by the PM status bits) to the accumulator.
MPYK *13-bit constant*	√	√	√	√	**Multiply Immediate** Multiply the contents of the T register (TMS320C2x/2xx) or TREG0 (TMS320C5x) by a signed 13-bit constant; place the result in the P register.
MPYS *dma* **MPYS** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Multiply and Subtract Previous Product** Multiply the contents of the T register (TMS320C2x/2xx) or TREG0 (TMS320C5x) by the contents of the addressed data-memory location; place the result in the P register. Subtract the previous product (shifted as specified by the PM status bits) from the accumulator.
MPYU *dma* **MPYU** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Multiply Unsigned** Multiply the unsigned contents of the T register (TMS320C2x/2xx) or TREG0 (TMS320C5x) by the unsigned contents of the addressed data-memory location; place the result in the P register.
NEG		√	√	√	**Negate Accumulator** Negate (2s complement) the contents of the accumulator.
NMI			√	√	**Nonmaskable Interrupt** Force the program counter to the nonmaskable interrupt vector location 24h. NMI has the same effect as a hardware nonmaskable interrupt.
NOP	√	√	√	√	**No Operation** Perform no operation.
NORM **NORM** {*ind*}		√ √	√ √	√ √	**Normalize Contents of Accumulator** Normalize a signed number in the accumulator.
OPL [**#***lk*,] *dma* **OPL** [**#***lk*,] {*ind*} [, *next ARP*]				√ √	**OR With DBMR or Long Immediate** If a long immediate is specified, OR it with the value at the specified data-memory location; otherwise, the second operand of the OR operation is the contents of the DBMR. The result is written back into the data-memory location previously holding the first operand.

Syntax	1x	2x	2xx	5x	Description
OR *dma*	√	√	√	√	**OR With Accumulator** TMS320C1x and TMS320C2x devices: OR the 16 LSBs of the accumulator with the contents of the addressed data-memory location. The 16 MSBs of the accumulator are ORed with 0s. TMS320C20x and TMS320C5x devices: OR the 16 LSBs of the accumulator or a 16-bit immediate value with the contents of the addressed data-memory location. If a shift is specified, left-shift before ORing. Low-order bits below and high-order bits above the shifted value are treated as 0s.
OR {*ind*} [, *next ARP*]	√	√	√	√	
OR **#***lk* [, *shift*]			√	√	
ORB				√	**OR ACCB With Accumulator** OR the contents of the ACCB with the contents of the accumulator. ORB places the result in the accumulator.
ORK **#***lk* [, *shift*]		√	√	√	**OR Immediate With Accumulator with Shift** OR a 16-bit immediate value with the contents of the accumulator. If a shift is specified, left-shift the constant before ORing. Low-order bits below and high-order bits above the shifted value are treated as 0s.
OUT *dma, PA*	√	√	√	√	**Output Data to Port** Write a 16-bit value from a data-memory location to the specified I/O port. TMS320C1x devices: The first cycle of this instruction places the port address onto address lines A2/PA2–A0/PA0. During the same cycle, $\overline{WE}$ goes low and the data word is placed on the data bus D15–D0. TMS320C2x, TMS320C20x, and TMS320C5x devices: The $\overline{IS}$ line goes low to indicate an I/O access; the $\overline{STRB}$, R/$\overline{W}$, and READY timings are the same as for an external data-memory write.
OUT {*ind*}, *PA* [, *next ARP*]	√	√	√	√	
PAC	√	√	√	√	**Load Accumulator With P Register** Load the contents of the P register into the accumulator. TMS320C2x, TMS320C20x, and TMS320C5x devices: Before the load, shift the P register as specified by the PM status bits.

Syntax	1x	2x	2xx	5x	Description
POP	√	√	√	√	**Pop Top of Stack to Low Accumulator** Copy the contents of the top of the stack into the 12 (TMS320C1x) or 16 (TMS320C2x/2xx/5x) LSBs of the accumulator and then pop the stack one level. The MSBs of the accumulator are zeroed.
POPD *dma* **POPD** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Pop Top of Stack to Data Memory** Transfer the value on the top of the stack into the addressed data-memory location and then pop the stack one level.
PSHD *dma* **PSHD** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Push Data-Memory Value Onto Stack** Copy the addressed data-memory location onto the top of the stack. The stack is pushed down one level before the value is copied.
PUSH	√	√	√	√	**Push Low Accumulator Onto Stack** Copy the contents of the 12 (TMS320C1x) or 16 (TMS320C2x/2xx/5x) LSBs of the accumulator onto the top of the hardware stack. The stack is pushed down one level before the value is copied.
RC		√	√	√	**Reset Carry Bit** Reset the C status bit to 0.
RET	√	√	√		**Return From Subroutine** Copy the contents of the top of the stack into the PC and pop the stack one level.
RET[*D*]				√	**Return From Subroutine With Optional Delay** Copy the contents of the top of the stack into the PC and pop the stack one level. If you specify a delayed branch (RETD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before the return.
RETC $cond_1$ [, $cond_2$] [, ...]			√		**Return Conditionally** If the specified conditions are met, RETC performs a standard return. Not all combinations of conditions are meaningful.

Syntax	1x	2x	2xx	5x	Description
RETC[*D*] *cond*$_1$ [, *cond*$_2$] [, ...]				√	**Return Conditionally With Optional Delay** If the specified conditions are met, RETC performs a standard return. Not all combinations of conditions are meaningful. If you specify a delayed branch (RETCD), the next two instruction words (two 1-word instructions or one 2-word instruction) are fetched and executed before the return.
RETE				√	**Enable Interrupts and Return From Interrupt** Copy the contents of the top of the stack into the PC and pop the stack one level. RETE automatically clears the global interrupt enable bit and pops the shadow registers (stored when the interrupt was taken) back into their corresponding strategic registers. The following registers are shadowed: ACC, ACCB, PREG, ST0, ST1, PMST, ARCR, INDX, TREG0, TREG1, TREG2.
RETI				√	**Return From Interrupt** Copy the contents of the top of the stack into the PC and pop the stack one level. RETI also pops the values in the shadow registers (stored when the interrupt was taken) back into their corresponding strategic registers. The following registers are shadowed: ACC, ACCB, PREG, ST0, ST1, PMST, ARCR, INDX, TREG0, TREG1, TREG2.
RFSM		√			**Reset Serial Port Frame Synchronization Mode** Reset the FSM status bit to 0.
RHM		√		√	**Reset Hold Mode** Reset the HM status bit to 0.
ROL		√	√	√	**Rotate Accumulator Left** Rotate the accumulator left one bit.
ROLB				√	**Rotate ACCB and Accumulator Left** Rotate the ACCB and the accumulator left by one bit; this results in a 65-bit rotation.
ROR		√	√	√	**Rotate Accumulator Right** Rotate the accumulator right one bit.
RORB				√	**Rotate ACCB and Accumulator Right** Rotate the ACCB and the accumulator right one bit; this results in a 65-bit rotation.

Syntax	1x	2x	2xx	5x	Description
ROVM	√	√	√	√	**Reset Overflow Mode** Reset the OVM status bit to 0; this disables overflow mode.
RPT *dma* **RPT** {*ind*} [, *next ARP*] **RPT #***k* **RPT #***lk*		√ √	√ √ √ √	√ √ √ √	**Repeat Next Instruction** TMS320C2x devices: Load the 8 LSBs of the addressed value into the RPTC; the instruction following RPT is executed the number of times indicated by RPTC + 1. TMS320C20x and TMS320C5x devices: Load the 8 LSBs of the addressed value or an 8-bit or 16-bit immediate value into the RPTC; the instruction following RPT is repeated *n* times, where *n* is RPTC+1.
RPTB *pma*				√	**Repeat Block** RPTB repeats a block of instructions the number of times specified by the memory-mapped BRCR without any penalty for looping. The BRCR must be loaded before RPTB is executed.
RPTK #*k*		√	√	√	**Repeat Instruction as Specified by Immediate Value** Load the 8-bit immediate value into the RPTC; the instruction following RPTK is executed the number of times indicated by RPTC + 1.
RPTZ #*lk*				√	**Repeat Preceded by Clearing the Accumulator and P Register** Clear the accumulator and product register and repeat the instruction following RPTZ *n* times, where *n* = *lk* +1.
RSXM		√	√	√	**Reset Sign-Extension Mode** Reset the SXM status bit to 0; this suppresses sign extension on shifted data values for the following arithmetic instructions: ADD, ADDT, ADLK, LAC, LACT, LALK, SBLK, SUB, and SUBT.
RTC		√	√	√	**Reset Test/Control Flag** Reset the TC status bit to 0.
RTXM		√			**Reset Serial Port Transmit Mode** Reset the TXM status bit to 0; this configures the serial port transmit section in a mode where it is controlled by an FSX.
RXF		√	√	√	**Reset External Flag** Reset XF pin and the XF status bit to 0.

Syntax	1x	2x	2xx	5x	Description
SACB				√	**Store Accumulator in ACCB** Copy the contents of the accumulator into the ACCB.
SACH *dma* [, *shift*] **SACH** {*ind*} [, *shift* [, *next ARP*]]	√ √	√ √	√ √	√ √	**Store High Accumulator With Shift** Copy the contents of the accumulator into a shifter. Shift the entire contents 0, 1, or 4 bits (TMS320C1x) or from 0 to 7 bits (TMS320C2x/2xx/5x), and then copy the 16 MSBs of the shifted value into the addressed data-memory location. The accumulator is not affected.
SACL *dma* **SACL** *dma* [, *shift*] **SACL** {*ind*} [, *shift* [, *next ARP*]]	√ √ 	 √ √	 √ √	 √ √	**Store Low Accumulator With Shift** TMS320C1x devices: Store the 16 LSBs of the accumulator into the addressed data-memory location. A shift value of 0 must be specified if the ARP is to be changed. TMS320C2x, TMS320C20x, and TMS320C5x devices: Store the 16 LSBs of the accumulator into the addressed data-memory location. If a shift is specified, shift the contents of the accumulator before storing. Shift values are 0, 1, or 4 bits (TMS320C20) or from 0 to 7 bits (TMS320C2x/2xx/5x).
SAMM *dma* **SAMM** {*ind*} [, *next ARP*]				√ √	**Store Accumulator in Memory-Mapped Register** Store the low word of the accumulator in the addressed memory-mapped register. The upper 9 bits of the data address are cleared, regardless of the current value of DP or the 9 MSBs of AR (ARP).
SAR *AR, dma* **SAR** *AR,* {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Store Auxiliary Register** Store the contents of the specified auxiliary register in the addressed data-memory location.
SATH				√	**Barrel-Shift Accumulator as Specified by T Register 1** If bit 4 of TREG1 is a 1, barrel-shift the accumulator right by 16 bits; otherwise, the accumulator is unaffected.
SATL				√	**Barrel-Shift Low Accumulator as Specified by T Register 1** Barrel-shift the accumulator right by the value specified in the 4 LSBs of TREG1.
SBB				√	**Subtract ACCB From Accumulator** Subtract the contents of the ACCB from the accumulator. The result is stored in the accumulator; the accumulator buffer is not affected.

Syntax	1x	2x	2xx	5x	Description
SBBB				√	**Subtract ACCB From Accumulator With Borrow** Subtract the contents of the ACCB and the logical inversion of the carry bit from the accumulator. The result is stored in the accumulator; the accumulator buffer is not affected. Clear the carry bit if the result generates a borrow.
SBLK #*lk* [*, shift*]		√	√	√	**Subtract From Accumulator Long Immediate With Shift** Subtract the immediate value from the accumulator. If a shift is specified, left shift the value before subtracting. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
SBRK #*k*		√	√	√	**Subtract From Auxiliary Register Short Immediate** Subtract the 8-bit immediate value from the designated auxiliary register.
SC		√	√	√	**Set Carry Bit** Set the C status bit to 1.
SETC *control bit*			√	√	**Set Control Bit** Set the specified control bit to a logic 1. Maskable interrupts are disabled immediately after the SETC instruction executes.
SFL		√	√	√	**Shift Accumulator Left** Shift the contents of the accumulator left one bit.
SFLB				√	**Shift ACCB and Accumulator Left** Shift the concatenation of the accumulator and the ACCB left one bit. The LSB of the ACCB is cleared to 0, and the MSB of the ACCB is shifted into the carry bit.
SFR		√	√	√	**Shift Accumulator Right** Shift the contents of the accumulator right one bit. If SXM = 1, SFR produces an arithmetic right shift. If SXM = 0, SFR produces a logic right shift.
SFRB				√	**Shift ACCB and Accumulator Right** Shift the concatenation of the accumulator and the ACCB right 1 bit. The LSB of the ACCB is shifted into the carry bit. If SXM = 1, SFRB produces an arithmetic right shift. If SXM = 0, SFRB produces a logic right shift.
SFSM		√			**Set Serial Port Frame Synchronization Mode** Set the FSM status bit to 1.

Syntax	1x	2x	2xx	5x	Description
SHM		√		√	**Set Hold Mode** Set the HM status bit to 1.
SMMR *dma*, **#***lk* **SMMR** {*ind*}, **#***lk* [, *next ARP*]				√ √	**Store Memory-Mapped Register** Store the memory-mapped register value, pointed at by the 7 LSBs of the data-memory address, into the long immediate addressed data-memory location. The 9 MSBs of the data-memory address of the memory-mapped register are cleared, regardless of the current value of DP or the upper 9 bits of AR(ARP).
SOVM	√	√	√	√	**Set Overflow Mode** Set the OVM status bit to 1; this enables overflow mode. (The ROVM instruction clears OVM.)
SPAC	√	√	√	√	**Subtract P Register From Accumulator** Subtract the contents of the P register from the contents of the accumulator. TMS320C2x, TMS320C20x, and TMS320C5x devices: Before the subtraction, shift the contents of the P register as specified by the PM status bits.
SPH *dma* **SPH** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Store High P Register** Store the high-order bits of the P register (shifted as specified by the PM status bits) at the addressed data-memory location.
SPL *dma* **SPL** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Store Low P Register** Store the low-order bits of the P register (shifted as specified by the PM status bits) at the addressed data-memory location.
SPLK **#***lk*, *dma* **SPLK** **#***lk*, {*ind*} [, *next ARP*]			√	√ √	**Store Parallel Long Immediate** Write a full 16-bit pattern into a memory location. The parallel logic unit (PLU) supports this bit manipulation independently of the ALU, so the accumulator is unaffected.
SPM *2-bit constant*		√	√	√	**Set P Register Output Shift Mode** Copy a 2-bit immediate value into the PM field of ST1. This controls shifting of the P register as shown below: PM = 00_2 Multiplier output is not shifted. PM = 01_2 Multiplier output is left shifted one place and zero filled. PM = 10_2 Multiplier output is left shifted four places and zero filled. PM = 11_2 Multiplier output is right shifted six places and sign extended; the LSBs are lost.

Syntax	1x	2x	2xx	5x	Description
SQRA *dma*		√	√	√	**Square and Accumulate Previous Product**
SQRA {*ind*} [, *next ARP*]		√	√	√	Add the contents of the P register (shifted as specified by the PM status bits) to the accumulator. Then load the contents of the addressed data-memory location into the T register (TMS320C2x/2xx) or TREG0 (TMS320C5x), square the value, and store the result in the P register.
SQRS *dma*		√	√	√	**Square and Subtract Previous Product**
SQRS {*ind*} [, *next ARP*]		√	√	√	Subtract the contents of the P register (shifted as specified by the PM status bits) to the accumulator. Then load the contents of the addressed data-memory location into the T register (TMS320C2x/2xx) or TREG0 (TMS320C5x), square the value, and store the result in the P register.
SST *dma*	√	√	√	√	**Store Status Register**
SST {*ind*} [, *next ARP*]	√	√	√	√	Store the contents of the ST (TMS320C1x) or ST0 (TMS320C2x/2xx/5x) in the addressed data-memory location.
SST #*n*, *dma*			√	√	**Store Status Register n**
SST #*n*, {*ind*} [, *next ARP*]			√	√	Store ST*n* in data memory.
SST1 *dma*		√	√	√	**Store Status Register ST1**
SST1 {*ind*} [, *next ARP*]		√	√	√	Store the contents of ST1 in the addressed data-memory location.
SSXM		√	√	√	**Set Sign-Extension Mode** Set the SXM status bit to 1; this enables sign extension.
STC		√	√	√	**Set Test/Control Flag** Set the TC flag to 1.
STXM		√			**Set Serial Port Transmit Mode** Set the TXM status bit to 1.

Syntax	1x	2x	2xx	5x	Description
SUB *dma* [, *shift*] **SUB** {*ind*} [, *shift* [, *next ARP*]] **SUB** #*k* **SUB** #*lk* [, $shift_2$]	√ √	√ √	√ √ √ √	√ √ √ √	**Subtract From Accumulator With Shift** TMS320C1x and TMS320C2x devices: Subtract the contents of the addressed data-memory location from the accumulator. If a shift is specified, left shift the value before subtracting. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1. TMS320C20x and TMS320C5x devices: Subtract the contents of the addressed data-memory location or an 8- or 16-bit constant from the accumulator. If a shift is specified, left shift the data before subtracting. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
SUBB *dma* **SUBB** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Subtract From Accumulator With Borrow** Subtract the contents of the addressed data-memory location and the value of the carry bit from the accumulator. The carry bit is affected in the normal manner.
SUBC *dma* **SUBC** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Conditional Subtract** Perform conditional subtraction. SUBC can be used for division.
SUBH *dma* **SUBH** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Subtract From High Accumulator** Subtract the contents of the addressed data-memory location from the 16 MSBs of the accumulator. The 16 LSBs of the accumulator are not affected.
SUBK #*k*		√	√	√	**Subtract From Accumulator Short Immediate** Subtract an 8-bit immediate value from the accumulator. The data is treated as an 8-bit positive number; sign extension is suppressed.
SUBS *dma* **SUBS** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Subtract From Low Accumulator With Sign Extension Suppressed** Subtract the contents of the addressed data-memory location from the accumulator. The data is treated as a 16-bit unsigned number; sign extension is suppressed.

Syntax	1x	2x	2xx	5x	Description
SUBT *dma* **SUBT** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Subtract From Accumulator With Shift Specified by T Register** Left shift the data-memory value as specified by the 4 LSBs of the T register (TMS320C2x/2xx) or TREG1 (TMS320C5x), and subtract the result from the accumulator. If a shift is specified, left shift the data-memory value before subtracting. During shifting, low-order bits are zero filled, and high-order bits are sign extended if SXM = 1.
SXF		√	√	√	**Set External Flag** Set the XF pin and the XF status bit to 1.
TBLR *dma* **TBLR** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Table Read** Transfer a word from program memory to a data-memory location. The program-memory address is in the 12 (TMS320C1x) or 16 (TMS320C2x/2xx/5x) LSBs of the accumulator.
TBLW *dma* **TBLW** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Table Write** Transfer a word from data-memory to a program-memory location. The program-memory address is in the 12 (TMS320C1x) or 16 (TMS320C2x/2xx/5x) LSBs of the accumulator.
TRAP		√	√	√	**Software Interrupt** The TRAP instruction is a software interrupt that transfers program control to program-memory address 30h (TMS320C2x) or 22h (TMS320C20x/5x) and pushes the PC + 1 onto the hardware stack. The instruction at address 30h or 22h may contain a branch instruction to transfer control to the TRAP routine. Putting the PC + 1 on the stack enables an RET instruction to pop the return PC.
XC *n*, $cond_1$ [, $cond_2$] [, ...]				√	**Execute Conditionally** Execute conditionally the next *n* instruction words where $1 \le n \le 2$. Not all combinations of conditions are meaningful.

Syntax	1x	2x	2xx	5x	Description
XOR *dma* **XOR** {*ind*} [, *next ARP*] **XOR #***lk* [, *shift*]	√ √ 	√ √ 	√ √ √	√ √ √	**Exclusive-OR With Accumulator** TMS320C1x and TMS320C2x devices: Exclusive-OR the contents of the addressed data-memory location with 16 LSBs of the accumulator. The MSBs are not affected. TMS320C20x and TMS320C5x devices: Exclusive-OR the contents of the addressed data-memory location or a 16-bit immediate value with the accumulator. If a shift is specified, left shift the value before performing the exclusive-OR operation. Low-order bits below and high-order bits above the shifted value are treated as 0s.
XORB				√	**Exclusive-OR of ACCB With Accumulator** Exclusive-OR the contents of the accumulator with the contents of the ACCB. The results are placed in the accumulator.
XORK #*lk* [, *shift*]		√	√	√	**Exclusive-OR Immediate With Accumulator With Shift** Exclusive-OR a 16-bit immediate value with the accumulator. If a shift is specified, left shift the value before performing the exclusive-OR operation. Low-order bits below and high-order bits above the shifted value are treated as 0s.
XPL [**#***lk*,] *dma* **XPL** [**#***lk*,] {*ind*} [, *next ARP*]				√ √	**Exclusive-OR of Long Immediate or DBMR With Addressed Data-Memory Value** If a long immediate value is specified, exclusive OR it with the addressed data-memory value; otherwise, exclusive OR the DBMR with the addressed data-memory value. Write the result back to the data-memory location. The accumulator is not affected.
ZAC	√	√	√	√	**Zero Accumulator** Clear the contents of the accumulator to 0.
ZALH *dma* **ZALH** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Zero Low Accumulator and Load High Accumulator** Clear the 16 LSBs of the accumulator to 0 and load the contents of the addressed data-memory location into the 16 MSBs of the accumulator.

Syntax	1x	2x	2xx	5x	Description
ZALR *dma* **ZALR** {*ind*} [, *next ARP*]		√ √	√ √	√ √	**Zero Low Accumulator, Load High Accumulator With Rounding** Load the contents of the addressed data-memory location into the 16 MSBs of the accumulator. The value is rounded by 1/2 LSB; that is, the 15 LSBs of the accumulator (0–14) are cleared and bit 15 is set to 1.
ZALS *dma* **ZALS** {*ind*} [, *next ARP*]	√ √	√ √	√ √	√ √	**Zero Accumulator, Load Low Accumulator With Sign Extension Suppressed** Load the contents of the addressed data-memory location into the 16 LSBs of the accumulator. The 16 MSBs are zeroed. The data is treated as a 16-bit unsigned number.
ZAP				√	**Zero the Accumulator and Product Register** The accumulator and product register are zeroed. The ZAP instruction speeds up the preparation for a repeat multiply/accumulate.
ZPR				√	**Zero the Product Register** The product register is cleared.

Appendix D

Program Examples

This appendix provides:

- ❑ A brief introduction to the process for generating executable program files.
- ❑ Sample programs for implementing simple routines and using interrupts, I/O pins, the timer, and the serial ports.

This appendix is not intended to teach you how to use the software development tools. The following documents cover these tools in detail:

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide (literature number SPRU018)

TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide (literature number SPRU024)

TMS320C2xx C Source Debugger User's Guide (literature number SPRU151)

For further information about ordering these documents, see *Related Documentation From Texas Instruments* on page vi of the Preface. For source code and examples, refer to the TI web site at **www.ti.com** and follow the *DSP* path to the *'C20x DSP.*

Topic **Page**

D.1 About These Program Examples D-2
D.2 Shared Program Code D-5
D.3 Task-Specific Program Code D-8
D.4 Introduction to Generating Bootloader Code D-23

D.1 About These Program Examples

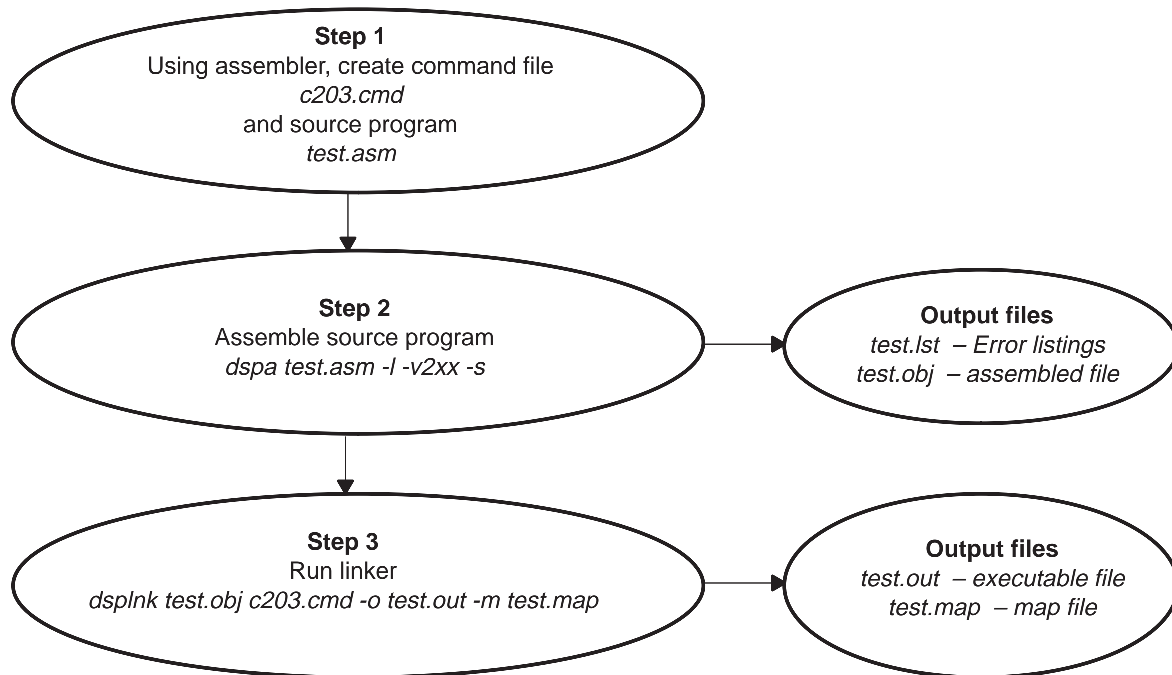
Figure D–1 illustrates the basic process for creating assembly language files and then generating executable files from them:

1) Use the 'C2xx assembler to create:
   - ❑ A command file (*c203.cmd* in the figure) that defines address ranges according to the architecture of the particular 'C2xx device
   - ❑ An assembly language program (*test.asm* in the figure)
2) Assemble the program. The command shown under Step 2 in the figure generates an object file and a file containing a listing of assembler errors encountered.
3) Use the linker to bring together the information in the object file and the command file and create an executable file (*test.out* in the figure). The command shown also generates a map file, which explains how the linker assigned the individual sections in the memory.

Note:

The procedure here applies to the PC™ development environment and is given only as an example.

Figure D–1. Procedure for Generating Executable Files



The program examples in section D.2 and section D.3 consist of code for shared files and task-specific files. Table D–1 describes the shared programs. Shared files contain code that is used by multiple task-specific files. The task-specific programs are described in Table D–2. Every task-specific file that uses the header files includes them by way of the *.copy* assembler directive:

```
.copy "init.h"
.copy "vector.h"
```

The assembler brings together the .h files and .asm file. The linker links assembled files according to the device architecture defined in the linker command file (c203.cmd).

Section D.4 contains an introduction to the procedure for using the assembler and linker to generate code for the bootloader. Program examples are also given in that section.

Table D–1. Shared Programs in This Appendix

Program	Functional Description	See ...
c203.cmd	Command file that defines size and placement of address blocks for the program, data, and I/O spaces	Example D–1, page D-5
init.h	Header file that declares space for variables and constants; declares initial values for variables; designates labels for the addresses of the control registers mapped to on-chip I/O space; contains comments that explain the functions of the control registers	Example D–2, page D-6
vector.h	Header file that fills the interrupt vector locations with branches to the corresponding interrupt service routines or with other values	Example D–3, page D-7

Table D–2. Task-Specific Programs in This Appendix

Program	Functional Description	See ...
delay.asm	Creates simple nested delay loops, measurable through XF and I/O pins	Example D–4, page D-8
timer.asm	Generates periodic timer interrupt, XF and I/O pins toggle at the interrupt rate	Example D–5, page D-9
intr1.asm	Causes XF pin to toggle at the rate of the interrupt signal on the $\overline{\text{INT1}}$ pin	Example D–6, page D-10
hold.asm	Explains the software logic for implementing a HOLD operation	Example D–7, page D-11
intr23.asm	Accepts an interrupt signal on $\overline{\text{INT2}}$ or $\overline{\text{INT3}}$. Toggles XF pin for each interrupt.	Example D–8, page D-12

Table D–2. Task-Specific Programs in This Appendix (Continued)

Program	Functional Description	See ...
uart.asm	Causes the asynchronous serial port to transmit a test message continuously at 1200 baud. Baud rate is 1200 at 50-ns cycle time.	Example D–9, page D-13
echo.asm	Echoes the character received by the asynchronous serial port at 1200 baud	Example D–10, page D-14
autobaud.asm	Causes the asynchronous serial port to lock on to the incoming baud rate and echoes the received character. The first character received should be *a* or *A*.	Example D–11, page D-16
bitio.asm	Toggles XF bit in response to delta interrupts and sends a character through the asynchronous serial port	Example D–12, page D-18
ssp.asm	Causes the synchronous serial port to send words in continuous mode with internal shift clock and frame synchronization	Example D–13, page D-20
ad55.asm	Implements simple loopback with a TLC320AD55C codec chip interfaced to the synchronous serial port	Example D–14, page D-21

D.2 Shared Program Code

Example D–1. Generic Command File (c203.cmd)

```
/* Title: c203.cmd                                              */
/* Generic command file for linking TMS320C20x assembler files */
/* input files:  *.obj files                                    */
/* output files: *.out file                                     */
/* Map files:    *.map file (optional)                          */
/* TMS320C20x architecture declaration for linker use           */

MEMORY
{

PAGE 0: /* PM - Program memory  */


EX1_PM     :ORIGIN=0H    ,  LENGTH=0FEFFH /* External program RAM */

B0_PM      :ORIGIN=0FF00H, LENGTH=0100H  /* BLOCK MAP IN CNF=1 */

PAGE 1: /* DM - Data memory    */

REGS       :ORIGIN=0H    ,  LENGTH=60H    /* MEM-MAPPED REGS            */
BLK_B2     :ORIGIN=60H   ,  LENGTH=20H    /* BLOCK B2                   */
BLK_B0     :ORIGIN=200H  ,  LENGTH=100H   /* BLOCK B0                   */
BLK_B1     :ORIGIN=300H  ,  LENGTH=100H   /* BLOCK B1                   */
EX1_DM     :ORIGIN=0800H ,  LENGTH=7800H  /* EXTERNAL DATA RAM          */
GM_DM      :ORIGIN=8000H ,  LENGTH=8000H  /* External DATA RAM AS GLOBAL*/

PAGE 2: /* I/O SPACE */
IO_IN      :ORIGIN=0FF00H,  LENGTH=0FFH   /* I/O MAPPED PERIPHERAL      */
IO_EX      :ORIGIN=0000H,   LENGTH=0FF00H /* EXT. I/O MAPPED PERIPHERAL */

}

SECTIONS
/* Linker directive to specify section placement in the memory map */

{
vectors    :{} > EX1_PM     PAGE 0        /*  Vectors at 0x0000         */
.text      :{} > EX1_PM     PAGE 0        /* .text placed after vectors */
.bss       :{} > EX1_DM     PAGE 1        /* .bss in 0x800 in DM        */
 new       :{} > BLK_B2     PAGE 1        /*  new in 0x0060 in DM       */
.data      :{} > 0x0370     PAGE 1        /* .data at 0x0370 in DM      */
}
```

Example D–2. Header File With I/O Register Declarations (init.h)

```
* File: init.h *
* Include file with I/O register declarations *

          .mmregs                    ; Include reserved words
          .bss   dmem,10             ; Undefined variables space
          .def   ini_d, start,codtx  ; Directive for symbol address
                                     ; generation in the current module
                                     ; –optional
ini_d:    .usect ”new”,10            ; Example of undefined variable space
                                     ; with the segment’s name as ”new”
          .data                      ; Example of including dummy constants
                                     ; –optional
          .word  055aah
          .word  0aa55h

* On–chip register equates
* CLKOUT
clk1      .set   0ffe8h
* INTERRUPT CONTROL
icr       .set   0ffech
* SYNC PORT
sdtr      .set   0fff0h
sspcr     .set   0fff1h
* UART
adtr      .set   0fff4h
aspcr     .set   0fff5h
iosr      .set   0fff6h
brd       .set   0fff7h
* TIMER
tcr       .set   0fff8h
prd       .set   0fff9h
tim       .set   0fffah
* WAIT STATES
wsgr      .set   0fffch

* Variables
rxbuf     .set   0300h
size      .set   00020h
del       .set   0010h
```

Example D–3. Header File With Interrupt Vector Declarations (vector.h)

```
* File:   vector.h                         *
* File    defines Interrupt vector labels  *
          .sect  ”vectors”
          b      start        ; reset vector – Jump to label start on reset
          b      inpt1        ; INT1 interrupt
          b      inpt23       ; INT2/INT3 interrupt
          b      timer        ; TINT  Timer interrupt
          b      codrx        ; RX_Sync interrupt
          b      codtx        ; TX_SYNC interrupt
          b      uart         ; TX/RX Uart port interrupt
                              ; Reserved and s/w interrupt vector locations
          .space 45*16        ; Directive for filling zeros in PM space
          .word  1,2,3,4,5    ; Example for constant loading
```

D.3 Task-Specific Program Code

Example D–4. Implementing Simple Delay Loops (delay.asm)

```
* File:      delay.asm                                                  *
* Function:  Delay loop. XF and I/O 3 pins toggle after each delay      *

           .title ”Delay routine”  ; Title
           .copy  ”init.h”         ; Variable and register declaration
           .copy  ”vector.h”       ; Vector label declaration
           .text

start:     clrc   cnf              ; Map block B0 to data memory
           ldp    #0h              ; set DP=0
           setc   INTM             ; Disable all interrupts
           splk   #0000h, 60h      ; Set zero wait states
           out    60h, wsgr
           splk   #0e00ch,60h      ; Define iosr for bit I/O in aspcr
           out    60h,aspcr
           lar    ar0,#del         ; Initialize ar0
           mar    *,ar7            ; Set ARP to ar7
           splk   #0008h,6eh       ; data for setting bit I/O 3
           splk   #0000h,6fh       ; data for clearing bit I/O 3
           splk   #0ffffh,60h      ; Inner repeat loop size
           lar    ar7,#del
loop:      clrc   xf               ; xf=0
           out    6fh,iosr         ; bit 3=0
dely1:     rpt    60h              ; @ 50ns, this loop gives 3.4 ms approx.
           nop
           banz   dely1,ar7        ; delay = 17*3.4 = 57.8 ms approx.
           lar    ar7,#del
           setc   xf               ; xf=1
           out    6eh,iosr         ; bit 3=1
dely2:     rpt    60h              ; @ 50ns, this loop gives 3.4 ms approx.
           nop
           banz   dely2,ar7        ; delay = 17*3.4 = 57.8 ms approx.
           lar    ar7,#del
           b      loop

inpt1:     ret                     ; Unused interrupts
inpt23:    ret                     ; have dummy returns for safety
timer:     ret
uart:      ret
codtx:     ret
codrx:     ret
           .end                    ; Assembler module end directive –optional
```

Example D–5. Testing and Using the Timer (timer.asm)

```
* File:   timer.asm                                          *
* Function: Timer test code                                  *
* PRD=0x00ff,TDDR=f @ 50ns, gives an interrupt interval=205us  *
* PRD=0xffff,TDDR=0 @ 50ns, gives an interrupt interval=3.27ms*
* Timer interval measurable on I/O 2,3 or xf pins            *

          .title ”Timer Test”     ; Title
          .copy  ”init.h”         ; Variable and register declaration
          .copy  ”vector.h”       ; Vector label declaration
          .text
start:    clrc   CNF              ; Map block B0 to data memory
          ldp    #0h              ; set DP=0
          setc   INTM             ; Disable all interrupts
          splk   #0000h,60h
          out    60h, wsgr        ; Set zero wait states
          splk   #0ffffh,ifr      ; clear interrupts
          splk   #0004h,imr       ; enable timer interrupt
          splk   #0e00ch, 60h     ; configure bit I/O I03 and IO2 as outputs
          out    60h, aspcr       ; set the aspcr for the above
          mar    *,ar1
          lar    ar1,#rxbuf
          splk   #0004h,61h       ; bit value to set I/O 2
          splk   #0008h,62h       ; bit value to set I/O 3
          out    61h,iosr         ; set the bit 2 = high, 3= zero
          splk   #0000h, 63h
          splk   #00ffh, 64h
          out    64h, prd         ; set PRD=0x00ffh
          out    63h, tim         ; set TIM=0x0000
          splk   #0c2fh, 64h      ; PSC, TDDR are zero, reload, restart
          out    64h, tcr
          clrc   intm
          clrc   xf
wait:     out    62h,iosr         ; set io2=0
          idle
          clrc   xf
          b      wait
timer:    setc   xf               ; xf =1
          in     68h,tcr          ; Read tcr,prd, tim regs.
          in     69h,prd
          in     6ah,tim
          out    61h,iosr         ; set io2=1
          clrc   intm
          ret
inpt1:    ret                     ; Unused interrupt routines
inpt23:   ret
codtx:    ret
codrx:    ret
uart:     ret
          .end                    ; Assembler module end directive –optional
```

Example D–6. Testing and Using Interrupt $\overline{INT1}$ (intr1.asm)

```
* File:   intr1.asm                                                    *
* Function: Interrupt test code                                        *
* For each INT1 interrupt XF,I/O pins IO3 and IO2 will toggle and      *
* transmit char 'c' through UART                                       *

          .title "Interrupt 1 Test"  ; Title
          .copy  "init.h"            ; Variable and register declaration
          .copy  "vector.h"          ; Vector label declaration
          .text
start:    clrc   CNF                 ; Map block B0 to data memory
          ldp    #0h                 ; set DP=0
          setc   INTM                ; Disable all interrupts
          splk   #0ffffh, ifr        ; clear interrupts
          splk   #0001h, imr         ; Enable int1 interrupts
          splk   #0010h, 60h
          out    60h,icr             ; Enable Intr1 in mode bit/ICR
          splk   #0000h, 60h
          out    60h, wsgr           ; Set zero wait states
          splk   #0e00ch, 60h        ; configure IO3 and IO2 as outputs
          out    60h, aspcr          ; set the aspcr for the above
          splk   #0411h, 60h         ; default baud rate 1200, for UART @50 ns
          out    60h,brd
          mar    *,ar1               ; Initialize AR pointer with AR1
          lar    ar1,#rxbuf
          lar    ar0,#size           ; set counter limit
          splk   #0004h,61h          ; set bit I/O 2
          splk   #0008h,62h          ; set bit I/O 3
          splk   #0063h,63h          ; set tx data
          clrc   INTM
          clrc   XF
wait:     out    61h,iosr            ; toggle IO2/3
          idle
          clrc   XF                  ; toggle xf
          b      wait

inpt1:    in     65h, icr            ; Read icr
          out    62h, iosr           ; toggle IO2/3
          out    65h, adtr           ; send icr value through UART to check
                                     ; interrupt source
          setc   XF                  ; toggle xf
          clrc   INTM
          ret

timer:    ret
inpt23:   ret
uart:     ret
codtx:    ret
codrx:    ret
          .end                       ; Assembler module end directive
                                     ; –optional
```

Example D–7. Implementing a HOLD Operation (hold.asm)

```
* File:      hold.asm                                         *
* Function:  HOLD test code                                   *
* Check for HOLDA toggle for HOLD requests in MODE 0          *
* Check for XF toggle on HOLD/INT1 requests in MODE 1         *

          .title ” HOLD Test ”   ; Title
          .mmregs
icr       .set   0FFECh          ; Interrupt control register in I/O space
icrshdw   .set   060h        ; scratch pad location

* Interrupt vectors
          .text
reset     B      main            ; 0–reset , Branch to main program on reset
int1h     B      int1_hold       ; 1–external interrupt 1 or HOLD
          .space 40*16

*********Interrupt service routine ISR for HOLD logic************************

main:     splk   #0001h,imr
          clrc   intm
wait:     b      wait
int1_hold:
          ; Perform any desired context save
          ldp    #0
          in     icrshdw, icr    ; save the contents of ICR register
          lacl   #010h           ; load ACC with mask for MODE bit
          and    icrshdw         ; Filter out all bits except MODE bit
          bcnd   int1,neq        ; Branch if MODE bit is 1, else in HOLD mode
          lacc   imr, 0          ; load ACC with interrupt mask register
          splk   #1, imr         ; mask all interrupts except interrupt1/HOLD
          idle                   ; enter HOLD mode, issues HOLDA
                                 ; and the busses will be in tristate
          splk   #1, ifr         ; Clear HOLD/INT1 flag to prevent
                                 ; re–entering HOLD mode
          sacl   imr             ; restore interrupt mask register
          ; Perform necessary context restore

          clrc   intm            ; enable all interrupts
          ret                    ; return from HOLD interrupt

int1:     nop                    ; Replace this with desired INT1 interrupt
          nop                    ; service routine
          setc   xf              ; Dummy toggle to check the loop entry
          clrc   xf              ; in MODE 1
          splk   #0001,ifr
          clrc   intm            ; enable all interrupts
          ret                    ; return from interrupts
```

Example D–8. Testing and Using Interrupts $\overline{INT2}$ and $\overline{INT3}$ (intr23.asm)

```
* File:      intr23.asm                                                *
* Function:  Interrupt test code                                       *
* Interrupt on INT2 or INT3 will toggle IO3 and IO2 bits               *
* and icr value copied in the Buffer @300                              *

          .title " Interrupt 2/3 Test"  ; Title
          .copy  "init.h"               ; Variable and register declaration
          .copy  "vector.h"             ; Vector label declaration
          .text
start:    clrc   CNF                    ; Map block B0 to data memory
          ldp #0h                       ; set DP=0
          setc   INTM                   ; Disable all interrupts
          splk   #0ffffh, ifr           ; clear interrupts
          splk   #0002h, imr            ; Enable int1 interrupts
          splk   #0003h, 60h
          out    60h, icr               ; Enable Int2 and 3 in ICR
          splk   #0000h, 60h
          out    60h, wsgr              ; Set zero wait states
          splk   #0e00ch, 60h           ; configure the I03 and IO2 as outputs
          out    60h, aspcr             ; set the aspcr for the above
          mar    *, ar1                 ; ARP=ar1
          lar    ar1, #rxbuf
          lar    ar0, #size             ; set counter limit
          splk   #0004h, 61h            ; set bit I/O 2
          splk   #0008h, 62h            ; set bit I/O 3
          splk   #0063h, 63h            ; set tx data
          clrc   intm
          clrc   xf
wait:     out    61h, iosr              ; toggle I/O 2
          idle
          clrc   xf                     ; toggle xf bit
          b      wait

inpt23:   in     65h, icr               ; Read icr
          in     *+, icr                ; Capture icr in buffer @300
          mar    *,ar0
          banz   skip, ar1
          lar    ar1, #rxbuf
          lar    ar0, #size
skip:     out    62h, iosr              ; toggle IO2/3
          setc   xf                     ; toggle xf
          out    65h, icr               ; clear interrupt 2/3 flag bit
          clrc   intm
          ret
timer:    ret
inpt1:    ret
uart:     ret
codtx:    ret
codrx:    ret
          .end                          ; Assembler module end directive
                                        ; -optional
```

Example D–9. Asynchronous Serial Port Transmission (uart.asm)

```
* File:   uart.asm                                                    *
* Function:  UART Test Code                                           *
* Continuously sends ''C203 UART is fine' at 1200 baud.               *

          .title " UART Test"     ; Title
          .copy  "init.h"         ; Variable and register declaration
          .copy  "vector.h"       ; Vector label declaration
          .text
start:    clrc   CNF              ; Map block B0 to data memory
          ldp    #0h              ; set DP=0
          setc   INTM             ; Disable all interrupts

* UART initialization *
          splk   #0ffffh,ifr         ; clear interrupts
          splk   #0000h,60h
          out    60h, wsgr           ; Set zero wait states
          splk   #0c180h,61h         ; reset the UART by writing 0
          out    61h, aspcr          ; 1 stop bit, tx interrupt, input i/o
          splk   #0e180h,61h         ; Enable the serial port
          out    61h,aspcr
          splk   #4fffh,62h
          out    62h,iosr            ; disable auto baud
          splk   #0411h, 63h         ; set baud rate =1200 @ 20-MHz CLKOUT1
          out    63h, brd
          splk   #20h,imr            ; enable UART interrupt
          mar    *,ar1               ; ARP=ar1
          lar    ar1,#rxbuf

* Load data at DM300                 ; 'c203 UART is fine!' – xmit data
          splk   #0063h,*+           ; ascii value for the above characters
          splk   #0032h,*+
          splk   #0030h,*+
          splk   #0033h,*+
          splk   #0020h,*+

          splk   #0055h,*+
          splk   #0041h,*+
          splk   #0052h,*+
          splk   #0054h,*+
          splk   #0020h,*+

          splk   #0069h,*+
          splk   #0073h,*+
          splk   #0020h,*+

          splk   #0066h,*+
          splk   #0069h,*+
          splk   #006eh,*+
          splk   #0065h,*+
          splk   #0020h,*+
          splk   #0021h,*+
          splk   #0021h,*+
          splk   #0020h,*+
```

Example D–9. Asynchronous Serial Port Transmission (uart.asm) (Continued)

```
          lar    ar1,#rxbuf
          lar    ar0, #20            ; load buffer size
          mar    *,ar1               ; load data pointer
          clrc   intm
wait:     clrc   xf                  ; toggle xf bit
          idle
          b      wait

uart:     setc   xf                  ; toggle xf bit
          splk   #0ffffh,67h
          out    *+,adtr             ; transmit character from data buffer@300
          mar    *,ar0
          banz   skip,ar1            ; check if size=0, and reload
          lar    ar1,#rxbuf
          lar    ar0,#20             ; set size = character length
skip:     splk   #0020h,ifr          ; Clear ifr bit
          clrc   intm
          ret
inpt1:    ret
inpt23:   ret
timer:    ret
codtx:    ret
codrx:    ret
          .end                       ; Assembler module end directive
                                     ; –optional
```

Example D–10. Loopback to Verify Transmissions of Asynchronous Serial Port (echo.asm)

```
* File:       echo.asm                                             *
* Function:   UART Test Code                                       *
*             Continuously echoes data received by UART at 1200 baud. *
*             Received data will be stored in the buffer @300      *

          .title ” UART/ASP loop back”   ; Title
          .copy  ”init.h”                ; Variable and register declaration
          .copy  ”vector.h”                 ; Vector label declaration
          .text
start:    clrc   CNF                     ; Map block B0 to data memory
          ldp    #0h                     ; set DP=0
          setc   INTM                    ; Disable all interrupts
```

Example D–10. Loopback to Verify Transmissions of Asynchronous Serial Port (echo.asm) (Continued)

```
* UART initialization *
          splk   #0ffffh,ifr             ; clear interrupts
          splk   #0000h,60h
          out    60h, wsgr               ; Set zero wait states
          splk   #0c080h,61h             ; reset the UART by writing 0
          out    61h, aspcr              ; 1 stop bit, rx interrupt, input i/o
          splk   #0e080h,61h
          out    61h,aspcr
          splk   #4fffh,62h
          out    62h,iosr                ; disable auto baud
          splk   #0411h, 63h             ; set baud rate =1200 @ 20MHz CLKOUT1
          out    63h, brd
          splk   #20h,imr                ; enable UART interrupt
          mar    *,ar1

* Load data at DM300
          lar    ar1,#rxbuf
          lar    ar0, #size              ; load buffer size
          mar    *,ar1                   ; load data pointer
          clrc intm

wait:     clrc   xf                      ; toggle xf bit
          idle
          b      wait

uart:     setc   xf                      ; toggle xf bit
                                         ; Check receive flag bit in iosr
          in     68h,iosr                ; load input status from iosr
          bit    68h,7                   ; bit 8 in the data
          bcnd   skip,ntc                ; IF DR=0 no echo, return
          in     *,adtr                  ; read and save at 300h
          out    *+,adtr                 ; echo
          mar    *,ar0
          banz   skip,ar1                ; check if size=0, and reload
          lar    ar1,#rxbuf
          lar    ar0,#size
skip:     splk   #0020h, ifr             ; Clear interrupt in ifr!
          clrc   intm
          ret
inpt1:    ret
inpt23:   ret
timer:    ret
codtx:    ret
codrx:    ret
          .end                           ; Assembler module end directive
                                         ; –optional
```

Example D–11. Testing and Using Automatic Baud-Rate Detection on Asynchronous Serial Port (autobaud.asm)

```
* File:       autobaud.asm                                             *
* Function:   UART,auto baud test                                      *
*             Locks to incoming baud rate if the first character       *
*             is ”A” or ”a” & continuously echoes data received        *
*             through the port.                                        *

* Once detection is complete, if the CAD and ADC bits are not          *
* disabled and the interrupt is enabled, the ISR will occur for        *
* all characters received and will change the baud setting again.      *

          .title ”Auto_baud detect”  ; Title
          .copy  ”init.h”            ; Variable and register declaration
          .copy  ”vector.h”          ; Vector label declaration
          .text

start:    clrc   CNF                 ; Map block B0 to data memory
          ldp    #0h                 ; set DP=0
          setc   INTM                ; Disable all interrupts

* UART initialization *
          splk   #0ffffh,ifr         ; clear interrupts
          splk   #0000h,60h
          out    60h, wsgr           ; Set zero wait states
          splk   #0c0a0h,61h         ; reset the UART by writing 0
          out    61h, aspcr          ; 1 stop bit, rx interrupt, input i/o
          splk   #0e0a0h,61h         ; CAD=1 enable
          out    61h,aspcr
          splk   #4fffh,62h          ; enable ADC bit
          out    62h,iosr            ; disable auto baud
          splk   #0000h, 63h         ; set baud rate =0000 @ 20-MHz CLKOUT1
          out    63h, brd
          splk   #20h,imr            ; enable UART interrupt
          mar    *,ar1
          lar    ar1,#rxbuf

* Load data at DM300
          lar    ar1,#rxbuf
          lar    ar0, #size          ; load buffer size
          mar    *,ar1               ; load data pointer
          clrc   intm
wait:     clrc   xf
          idle
          b      wait
```

Example D–11. Testing and Using Automatic Baud-Rate Detection on Asynchronous Serial Port (autobaud.asm) (Continued)

```
uart:
          setc   xf
          in     68h,iosr              ; load input status from iosr
          bit    68h,1                 ; check if auto baud bit is set
          bcnd   rcv,ntc               ; branch normal receive
          splk   #4fffh,67h            ; clear ADC
          out    67h,iosr
          splk   #0e080h,67h
          out    67h, aspcr            ; Disable CAD bit/auto baud
rcv:      in     68h,iosr              ; check for DR bit
          bit    68h,7                 ; bit 8 in the data
          bcnd   skip,ntc              ; IF DR=0 no echo, return
          in     *,adtr                ; read and save at 300h
          out    *+,adtr               ; echo
          mar    *,ar0
          banz   skip,ar1              ; check if size=0, and reload
          lar    ar1,#rxbuf
          lar    ar0,#size
skip:     splk   #0020h,ifr            ; Clear ifr
          clrc   intm
          ret
inpt1:    ret
inpt23:   ret
timer:    ret
codtx:    ret
codrx:    ret
          .end                         ; Assembler module end directive
                                       ; –optional
```

Example D–12. Testing and Using Asynchronous Serial Port Delta Interrupts (bitio.asm)

```
* File:       bitio.asm                                            *
* Function:   Delta interrupt  test code                           *
*             Accepts delta interrupt on IO pins 3 and 2           *
*             If bit level changes on bit 7, send character 'c'    *
*             through UART & toggle xf pin.                        *
*             If bit level changes on bit 6, send character 'i'    *
*             through UART & toggle xf pin.                        *
*             The delta bits are cleared after interrupt service   *

          .title "BIT IO Interrupt Test" ; Title
          .copy  "init.h"                ; Variable and register declaration
          .copy  "vector.h"              ; Vector label declaration
          .text
start:    clrc   CNF                     ; Map block B0 to data memory
          ldp    #0h                     ; set DP=0
          setc   INTM                    ; Disable all interrupts

* UART initialization *
          splk   #0ffffh,ifr             ; clear interrupts
          splk   #0000h,60h
          out    60h, wsgr               ; Set zero wait states
          splk   #0c200h,61h             ; reset the UART by writing 0
          out    61h, aspcr              ; 1 stop bit, Delta interrupt,
                                         ; input i/o
          splk   #0e200h,61h
          out    61h,aspcr
          splk   #4fffh,62h
          out    62h,iosr                ; disable auto baud
          splk   #0411h, 63h             ; set baud rate =1200 @ 20-MHz CLKOUT1
          out    63h, brd
          splk   #20h,imr                ; enable UART interrupt
          splk   #0063h,65h              ; transmit value = 0063h ='c'
          splk   #0069h,67h              ; transmit value = 0063h ='i'
          mar    *,ar1
          lar    ar1,#rxbuf

* Load data at DM300 *
          lar    ar1,#rxbuf
          lar    ar0, #size              ; load buffer size
          mar    *,ar1                   ; load data pointer
          clrc   intm                    ; disable interrupts for polling
wait:
          idle
          b      wait
```

Example D–12. Testing and Using Asynchronous Serial Port Delta Interrupts(bitio.asm) (Continued)

```
uart:     setc   xf                    ; toggle xf bit
          in     68h,iosr              ; Bit i/o check
          bit    68h,8                 ; bit address 7 I/O 3 BIT IS SET?
                                       ; required bit place = complement 7 !
          bcnd   poll,ntc              ; NO then check FOR I/O 2
          clrc   tc
          out    65h, adtr             ; transmit 63h ='c'
          splk   #0080h,6bh            ; reset delta bit
          out    6bh,iosr              ; THE DELTA INTERRUPTS WILL BE ALWAYS
                                       ; COMING IF THIS IS NOT CLEARED!!!
          clrc   xf                    ; clear xf bit
          splk   #20h,ifr              ; clear ifr bits
          clrc   intm
          ret
poll:     in     68h,iosr
          bit    68h,9                 ; bit address 6 I/O 2 bit is set?
          bcnd   poll1,ntc
          clrc   tc
          out    67h, adtr             ; if set transmit 69h = 'i'
          splk   #0040h,6bh            ; reset delta bit
          out    6bh,iosr
poll1:    clrc   xf                    ; clear xf bit
          splk   #20h,ifr              ; clear ifr bits
          clrc   intm
          ret
inpt1:    ret
inpt23:   ret
timer:    ret
codtx:    ret
codrx:    ret
          .end                         ; Assembler module end directive
                                       ; –optional
```

Example D–13. Synchronous Serial Port Continuous Mode Transmission (ssp.asm)

```
* File:       ssp.asm                                                  *
* Function:   Continuous transmit in CONTINUOUS mode                   *
*             Internal shift clock and frame sync                      *
*             Transmit FIFO level is set to 4                          *

          .title ”SSP Continuous mode”   ; Title
          .copy  ”init.h”                ; Variable and register declaration
          .copy  ”vector.h”              ; Vector label declaration
          .text
start:    clrc   cnf                     ; Map block B0 to data memory
          ldp    #0h                     ; set DP=0
          setc   INTM                    ; Disable all interrupts
          splk   #0000h, 60h             ; Set zero wait states
          out    60h, wsgr
          splk   #0cc0ch,60h             ; reset the serial port by writing
          out    60h, sspcr              ; zeros at NOR/RES
          splk   #0cc3ch,60h             ; enable Sync port, 4 word fifo,
          out    60h,sspcr               ; internal clocks, Continuous mode
                                         ; Use sspcr= #0cc3eh for Burst mode
          splk   #1717h,61h              ; dummy data for tx
          splk   #7171h,63h
          splk   #0aa55h,64h
          splk   #55aah,62h              ; transmit 55aah on tx
          splk   #10h,imr                ; enable xinit interrupt
          clrc   intm                    ; enable INTM
          out    62h,sdtr                ; Xmit once to start
          out    61h,sdtr                ; transmit interrupts
          out    63h,sdtr
          out    64h,sdtr

loop:     clrc   xf                      ; clear xf flag
          idle
          b      loop

codtx:    setc   xf                      ; set xf bit
          out    62h,sdtr                ; transmit 0x55aah again
          out    61h,sdtr                ; transmit 1717h
          out    63h,sdtr                ; transmit 7171h
          out    64h,sdtr                ; transmit aa55h
          splk   #0010h, ifr             ; clear ifr flag
          clrc   intm
          ret
codrx:    ret
inpt1:    ret
inpt23:   ret
timer:    ret
uart:     ret
          .end                           ; Assembler module end directive
                                         ; –optional
```

Example D–14. Using Synchronous Serial Port With Codec Device (ad55.asm)

```
* File:       ad55.asm                                              *
* Function:   Burst mode simple loop back on AD55 CODEC             *
*             CODEC master clock 10 MHz                             *
*             Simple I/O at 9.6-kHz sampling                        *

          .title ”AD55 codec simple I/O”; Title
          .copy  ”init.h”              ; Variable and register declaration
          .copy  ”vector.h”            ; Vector label declaration
          .text
start:    clrc   cnf                   ; Map block B0 to data memory
          ldp    #0h                   ; set DP=0
          setc   intm                  ; Disable all interrupts
          splk   #0000h, 60h           ; Set zero wait states
          out    60h,wsgr
          splk   #0c002h,60h           ; Initialize SSP
          out    60h, sspcr            ; reset the serial port by writing
          splk   #0c032h,60h           ; zeros to reset bits,
          out    60h,sspcr             ; enable Sync port, 1 word fifo,
                                       ; CLX/FSR as inputs. Burst mode

main:     splk   #08h,imr              ; enable RINT interrupt
          splk   #0ffffh, ifr          ; reset ifr flags
          mar    *,ar1                 ; load ar1 with rx buffer
          lar    ar1, #rxbuf
          lar    ar0, #size
* 0    0  R/W’   reg_add   data        ; AD55 command reg. bits
*D15   14 13     12 – 8    7–0
          splk   #0000h, 60h           ; reg0 nop
          splk   #0304h, 61h           ; reg1 9.6khz sampling
          splk   #0200h, 62h           ; default data 00
          splk   #0301h, 63h           ; default data 01
          splk   #0401h, 64h           ; default data 01
          splk   #0508h, 65h           ; default data 08
          splk   #0001h, 66h           ; secondary comm. request data
          out    66h,sdtr              ; request sec. comm.
          out    61h,sdtr              ; send reg1 data for 9.6-Khz sampling
          out    60h,sdtr              ; send 0x0000 after programming
          clrc   intm                  ; Enable SSP interrupts
loop:     clrc   xf                    ; clear xf flag
          idle                         ; Wait for SSP interrupt
          b      loop
```

Example D–14. Using Synchronous Serial Port With Codec Device (ad55.asm) (Continued)

```
codtx:     splk   #0010h, ifr              ; clear tx intr flag
           clrc   intm
           ret

codrx:     setc   xf                       ; toggle xf bit
           in     *,sdtr                   ; Read ADC value
           lacc   *+,0                     ; Make LSB zero
           and    #0fffeh,0                ; to avoid secondary
           sacl   6ah,0                    ; request for codec
           out    6ah,sdtr                 ; Send ADC value to DAC
           mar    *,ar0
           banz   skip,ar1                 ; Check buffer limits
           lar    ar1,#rxbuf
           lar    ar0,#size
skip:      splk   #0008h, ifr              ; Clear ifr flag
           clrc   intm
           ret
inpt1:     ret
inpt23:    ret
timer:     ret
uart:      ret
           .end                            ; Assembler module end directive
                                           ; –optional
```

D.4 Introduction to Generating Bootloader Code

The ’C2xx on-chip bootloader boots software from an 8-bit external EPROM to a 16-bit external RAM at reset. This section introduces to the procedure for using Texas Instruments development tools to generate the code that will be loaded into the EPROM.

Note:

The procedure in this section is given only as an example. This procedure may have to be modified to suit different applications.

For more details, refer to the *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User’s Guide* (literature number SPRU018).

The process for generating bootloader code uses these basic steps:

1) Write the following code by using the TMS320C1x/C2x/C2xx/C5x assembler:

   - The code that you wish to have loaded into the EPROM. Program code is listed after a .text assembler directive (see any of the programs in section D.3).
   - A linker command file that defines the architecture of the particular ’C2xx device being used. Example D–15 shows a command file for the ’C203. Note that the file declares the .text section at 0000h. This is necessary because the bootloader transfers the code to the external RAM beginning at address 0000h.

2) Assemble the code. Use the –v2xx option (for ’C2xx assembly) in the assemble command.

3) Link the assembled file with the command file by using the TMS320C1x/C2x/C2xx/C5x linker.

4) Write a hex conversion command file (an ASCII file) that contains options and directives for the TMS320C1x/C2x/C2xx/C5x hex conversion utility. Example D–16 shows such a file.

5) Use the hex conversion command file with the hex conversion utility to generate the boot code in an ASCII hexadecimal format suitable for loading into an EPROM programmer. The command file in Example D–16 selects the Intel™ format.

Example D–15. Linker Command File

```
MEMORY
{
PAGE 0:   /* PM – Program memory  */
EX1_PM    :ORIGIN=0H    ,  LENGTH=0FEFFH /* External program RAM */
B0_PM     :ORIGIN=0FF00H,  LENGTH=0100H  /* BLOCK MAP IN CNF=1 */
PAGE 1:   /* DM – Data memory    */
REGS      :ORIGIN=0H    ,  LENGTH=60H    /* MEM–MAPPED REGS */
BLK_B2    :ORIGIN=60H   ,  LENGTH=20H    /* BLOCK B2  */
BLK_B0    :ORIGIN=200H  ,  LENGTH=100H   /* BLOCK B0, */
BLK_B1    :ORIGIN=300H  ,  LENGTH=100H   /* BLOCK B1  */
EX1_DM    :ORIGIN=0800H ,  LENGTH=7800H  /* EXTERNAL DATA RAM */
GM_DM     :ORIGIN=8000H ,  LENGTH=8000H  /* External DATA RAM AS GLOBAL */
PAGE 2:   /* I/O SPACE */
IO_IN     :ORIGIN=0FF00H,  LENGTH=0FFH   /* I/O MAPPED PERIPHERAL     */
IO_EX     :ORIGIN=0000H ,  LENGTH=0FF00H /* EXT. I/O MAPPED PERIPHERAL */
}

SECTIONS
/* Linker directive to specify section placement in the memory map */
{

   .text  :{}    > EX1_PM  PAGE 0
}
```

Example D–16. Hex Conversion Utility Command File

```
      dsphex boot.cmd
/* boot.cmd file an example */
test.out            /* File for boot code in COFF format*/
–i                  /* option to generate Intel hex format */
–o test.i0          /* Name of the output file */
–byte               /* 16–bit code is converted into byte */
                    /* stack to suit 8–bit ROM. */
–order MS           /* The byte order is higher byte first followed by */
                    /* lower order byte                                */
–memwidth 8
–romwidth 8
–boot
SECTIONS
{  .text:boot }
```

Appendix E

Submitting ROM Codes to TI

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:

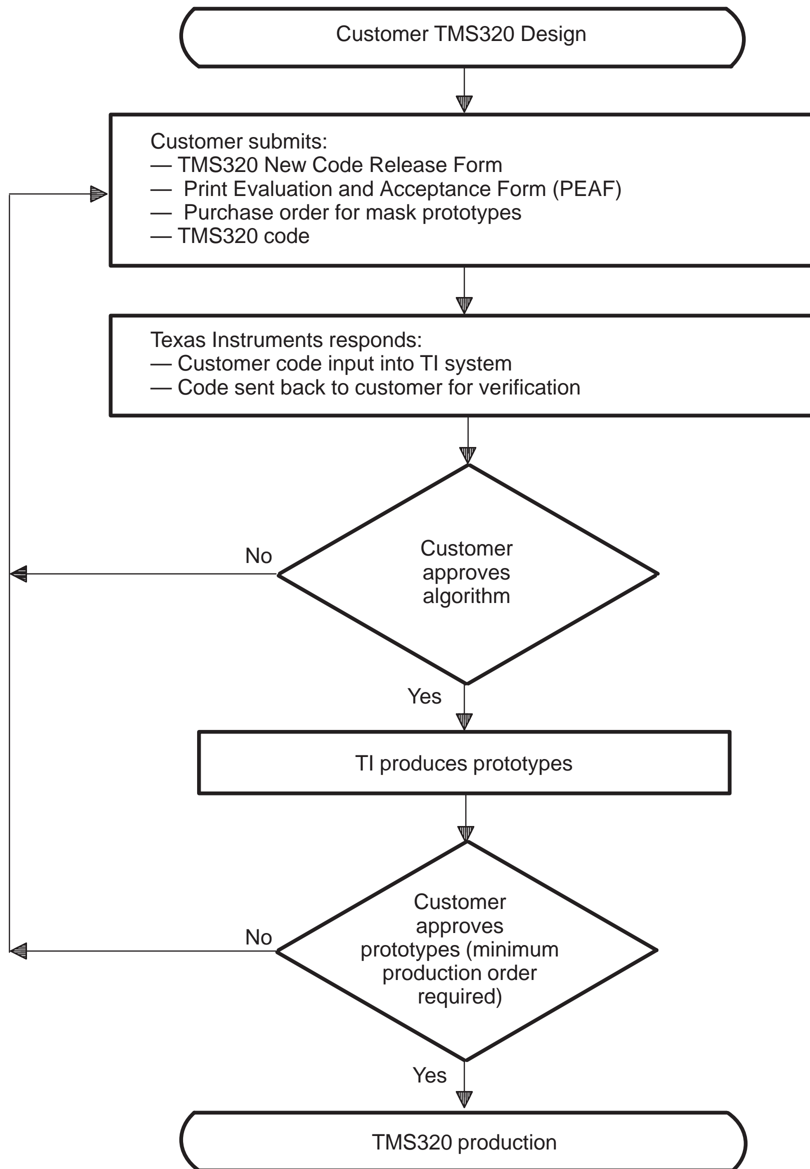
- Greater memory expansion
- Lower system cost
- Less hardware and wiring
- Smaller PCB

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.

Figure E–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 will be deleted from the TI system one year after the final delivery.

Figure E–1. TMS320 ROM Code Submittal Flow Chart



The TMS320 ROM code may be submitted in one of the following forms:

- Attachment to an email
- 3-1/2-in floppy: COFF format from macro-assembler/linker (preferred)

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 F

Design Considerations for Using XDS510 Emulator

This appendix assists you in meeting the design requirements of the Texas Instruments XDS510 emulator with respect to IEEE-1149.1 designs and discusses the XDS510 cable (manufacturing part number 2617698-0001). This cable is identified by a label on the cable pod marked *JTAG 3/5V* and supports both standard 3-V and 5-V target system power inputs.

The term *JTAG,* as used in this book, refers to TI scan-based emulation, which is based on the IEEE 1149.1 standard.

For more information concerning the IEEE 1149.1 standard, 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**

F.1 Designing Your Target System's Emulator Connector (14-Pin Header) F-2

F.2 Bus Protocol F-4

F.3 Emulator Cable Pod F-5

F.4 Emulator Cable Pod Signal Timing F-6

F.5 Emulation Timing Calculations F-7

F.6 Connections Between the Emulator and the Target System F-10

F.7 Physical Dimensions for the 14-Pin Emulator Connector F-14

F.8 Emulation Design Considerations F-16

F.1 Designing Your Target System's Emulator Connector (14-Pin Header)

JTAG target devices support emulation through a dedicated emulation port. This port is accessed directly by the emulator and provides emulation functions that are a superset of those specified by IEEE 1149.1. To communicate with the emulator, *your target system must have a 14-pin header* (two rows of seven pins) with the connections that are shown in Figure F–1. Table F–1 describes the emulation signals.

Although you can use other headers, the recommended unshrouded, straight header has these DuPont connector systems part numbers:

- ❑ 65610–114
- ❑ 65611–114
- ❑ 67996–114
- ❑ 67997–114

Figure F–1. 14-Pin Header Signals and Header Dimensions

Signal	Pin	Pin	Signal
TMS	1	2	$\overline{TRST}$
TDI	3	4	GND
PD (V_{CC})	5	6	no pin (key)†
TDO	7	8	GND
TCK_RET	9	10	GND
TCK	11	12	GND
EMU0	13	14	EMU1

Header Dimensions:
Pin-to-pin spacing, 0.100 in. (X,Y)
Pin width, 0.025-in. square post
Pin length, 0.235-in. nominal

† While the corresponding female position on the cable connector is plugged to prevent improper connection, the cable lead for pin 6 is present in the cable and is grounded, as shown in the schematics and wiring diagrams in this appendix.

Table F–1. 14-Pin Header Signal Descriptions

Signal	Description	Emulator† State	Target† State
EMU0	Emulation pin 0	I	I/O
EMU1	Emulation pin 1	I	I/O
GND	Ground		
PD(V_{CC})	Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD should be tied to V_{CC} in the target system.	I	O
TCK	Test clock. TCK is a 10.368-MHz clock source from the emulation cable pod. This signal can be used to drive the system test clock.	O	I
TCK_RET	Test clock return. Test clock input to the emulator. May be a buffered or unbuffered version of TCK.	I	O
TDI	Test data input	O	I
TDO	Test data output	I	O
TMS	Test mode select	O	I
$\overline{\text{TRST}}$‡	Test reset	O	I

† I = input; O = output

‡ Do not use pullup resistors on $\overline{\text{TRST}}$: it has an internal pulldown device. In a low-noise environment, $\overline{\text{TRST}}$ can be left floating. In a high-noise environment, an additional pulldown resistor may be needed. (The size of this resistor should be based on electrical current considerations.)

F.2 Bus Protocol

The IEEE 1149.1 specification covers the requirements for the test access port (TAP) bus slave devices 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 these 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.

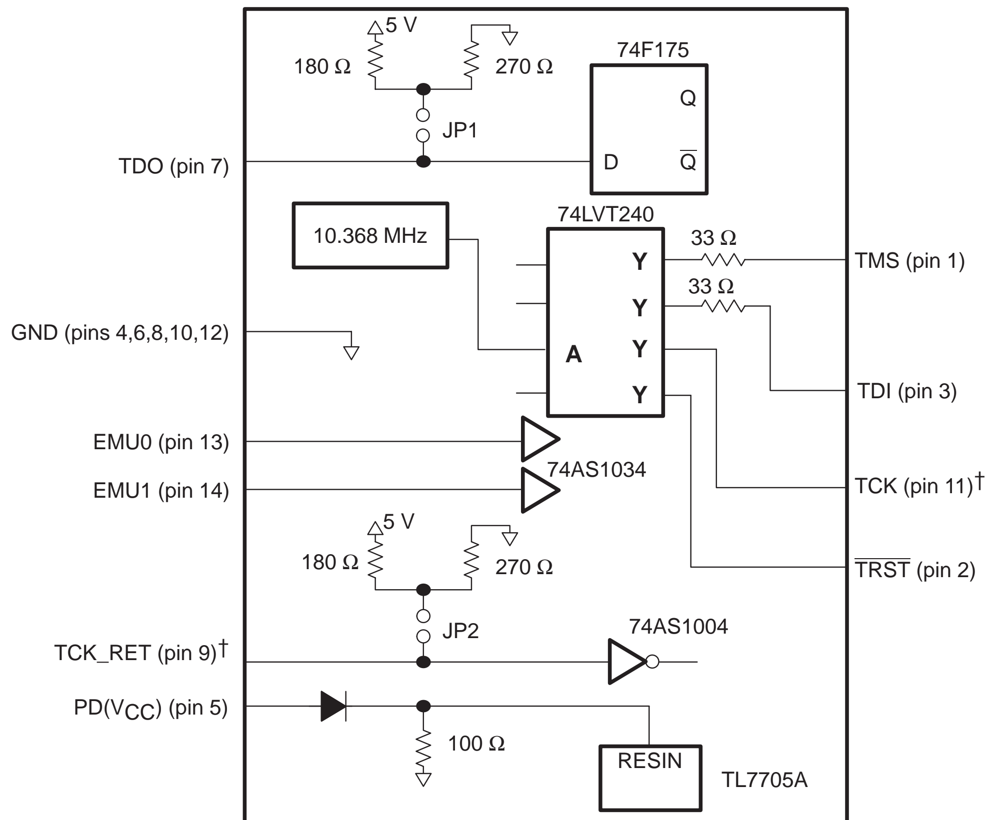
The IEEE 1149.1 specification does not provide rules for bus master (emulator) devices. Instead, it states that the device expects a bus master to provide bus slave compatible timings. The XDS510 provides timings that meet the bus slave rules.

F.3 Emulator Cable Pod

Figure F–2 shows a portion of the emulator cable pod. The functional features of the 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 74LVT240 device. Because of the high-current drive (32-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 1149.1 bus slave device timing rules.
- TMS and TDI are series-terminated to reduce signal reflections.
- A 10.368-MHz test clock source is provided. You can also provide your own test clock for greater flexibility.

Figure F–2. Emulator Cable Pod Interface



† The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

F.4 Emulator Cable Pod Signal Timing

Figure F–3 shows the signal timings for the emulator cable pod. Table F–2 defines the timing parameters illustrated in the figure. These timing parameters are calculated from values specified in the standard data sheets for the emulator and 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 F–3. Emulator Cable Pod Timings

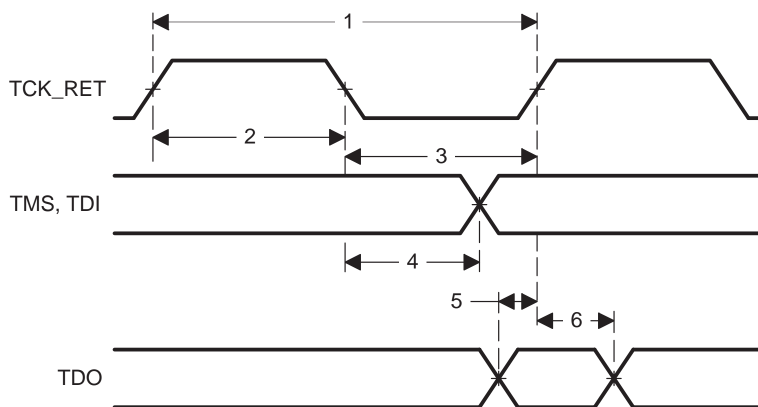


Table F–2. Emulator Cable Pod Timing Parameters

No.	Parameter	Description	Min	Max	Unit
1	$t_{c(TCK)}$	Cycle time, TCK_RET	35	200	ns
2	$t_{w(TCKH)}$	Pulse duration, TCK_RET high	15		ns
3	$t_{w(TCKL)}$	Pulse duration, TCK_RET low	15		ns
4	$t_{d(TMS)}$	Delay time, TMS or TDI valid for TCK_RET low	6	20	ns
5	$t_{su(TDO)}$	Setup time, TDO to TCK_RET high	3		ns
6	$t_{h(TDO)}$	Hold time, TDO from TCK_RET high	12		ns

F.5 Emulation Timing Calculations

Example F–1 and Example F–2 help you calculate emulation timings in your system. For actual target timing parameters, see the appropriate data sheet for the device you are emulating.

The examples use the following assumptions:

$t_{su(TTMS)}$	Setup time, target TMS or TDI to TCK high	10 ns
$t_{d(TTDO)}$	Delay time, target TDO from TCK low	15 ns
$t_{d(bufmax)}$	Delay time, target buffer maximum	10 ns
$t_{d(bufmin)}$	Delay time, target buffer minimum	1 ns
$t_{bufskew}$	Skew time, target buffer between two devices in the same package: $[t_{d(bufmax)} - t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{TCKfactor}$	Duty cycle, assume a 40/60% duty cycle clock	0.4 (40%)

Also, the examples use the following values from Table F–2 on page F-6:

$t_{d(TMSmax)}$	Delay time, emulator TMS or TDI from TCK_RET low, maximum	20 ns
$t_{su(TDOmin)}$	Setup time, TDO to emulator TCK_RET high, minimum	3 ns

There are two key timing paths to consider in the emulation design:

- ❑ The TCK_RET-to-TMS or TDI path, called $t_{pd(TCK_RET-TMS/TDI)}$ (propagation delay time)
- ❑ The TCK_RET-to-TDO path, called $t_{pd(TCK_RET-TDO)}$

In the examples, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Example F–1. Key Timing for a Single-Processor System Without Buffers

$$t_{pd(TCK_RET\text{-}TMS/TDI)} = \frac{\left[t_{d(TMSmax)} + t_{su(TTMS)}\right]}{t_{TCKfactor}}$$

$$= \frac{(20\text{ ns} + 10\text{ ns})}{0.4}$$

$$= 75\text{ ns, or } 13.3\text{ MHz}$$

$$t_{pd(TCK_RET\text{–}TDO)} = \frac{\left[t_{d(TTDO)} + t_{su(TDOmin)}\right]}{t_{TCKfactor}}$$

$$= \frac{(15\text{ ns} + 3\text{ ns})}{0.4}$$

$$= 45\text{ ns, or } 22.2\text{ MHz}$$

In this case, because the TCK_RET-to-TMS/TDI path requires more time to complete, it is the limiting factor.

Example F–2. Key Timing for a Single- or Multiple-Processor System With Buffered Input and Output

$$t_{pd(TCK_RET\text{-}TMS/TDI)} = \frac{\left[t_{d(TMSmax)} + t_{su(TTMS)} + t_{bufskew}\right]}{t_{TCKfactor}}$$

$$= \frac{(20\text{ ns} + 10\text{ ns} + 1.35\text{ ns})}{0.4}$$

$$= 78.4\text{ ns, or } 12.7\text{ MHz}$$

$$t_{pd(TCK_RET\text{–}TDO)} = \frac{\left[t_{d(TTDO)} + t_{su(TDOmin)} + t_{d(bufmax)}\right]}{t_{TCKfactor}}$$

$$= \frac{(15\text{ ns} + 3\text{ ns} + 10\text{ ns})}{0.4}$$

$$= 70\text{ ns, or } 14.3\text{ MHz}$$

In this case also, because the TCK_RET-to-TMS/TDI path requires more time to complete, it 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 parameter is called rise time, t_r. This can be calculated as follows:

$$
\begin{aligned}
t_r &= 5(R_{pullup} \times N_{devices} \times C_{load_per_device}) \\
&= 5(4.7\ k\Omega \times 16 \times 15\ pF) \\
&= 5(4.7 \times 10^{3}\ \Omega \times 16 \times 15 = no^{-12}\ F) \\
&= 5(1128 \times 10^{-9}) \\
&= 5.64\ \mu s
\end{aligned}
$$

F.6 Connections Between the Emulator and the Target System

It is extremely important to provide high-quality signals between the emulator and the JTAG target system. You must supply the correct signal buffering, test clock inputs, and multiple processor interconnections to ensure proper emulator and target system operation.

Signals applied to the EMU0 and EMU1 pins on the JTAG target device can be either input or output. In general, these two pins are used as both input and output in multiprocessor systems to handle global run/stop operations. EMU0 and EMU1 signals are applied only as inputs to the XDS510 emulator header.

F.6.1 Buffering Signals

If the distance between the emulation header and the JTAG target device is greater than 6 inches, the emulation signals must be buffered. If the distance is less than 6 inches, no buffering is necessary. Figure F–4 shows the simpler, no-buffering situation.

The distance between the header and the JTAG target device must be no more than 6 inches. The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

Figure F–4. Emulator Connections Without Signal Buffering

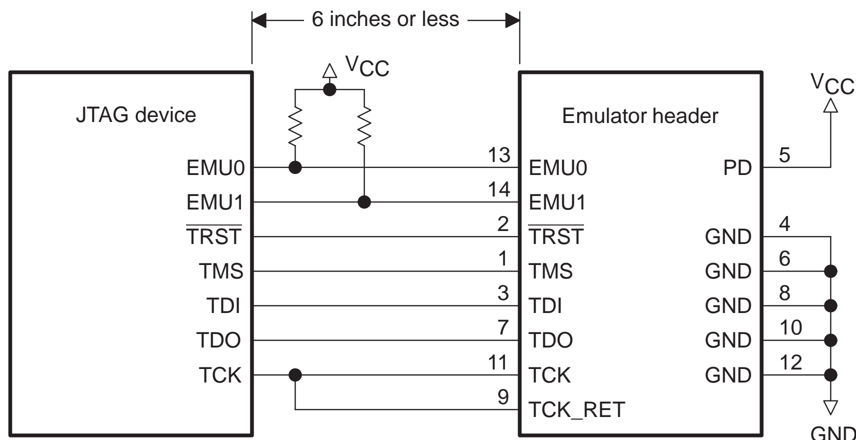
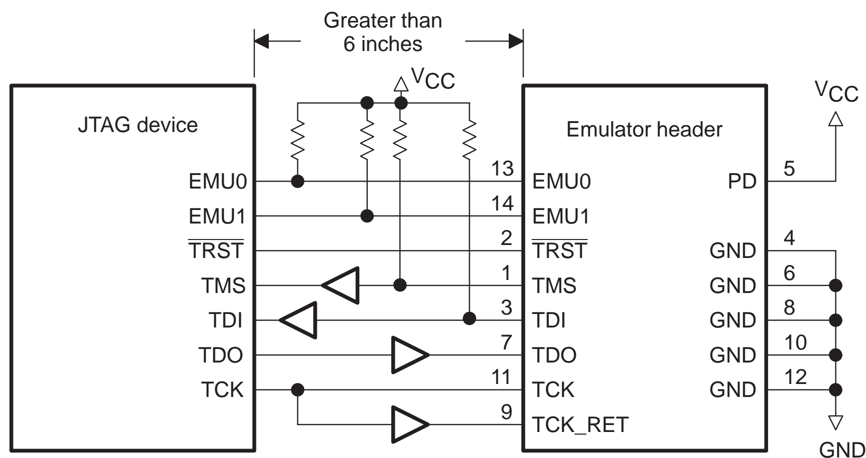


Figure F–5 shows the connections necessary for buffered transmission signals. The distance between the emulation header and the processor is greater than 6 inches. Emulation signals TMS, TDI, TDO, and TCK_RET are buffered through the same device package.

Figure F–5. Emulator Connections With Signal Buffering



The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

The input buffers for TMS and TDI should have pullup resistors connected to V_{CC} to hold these signals at a known value when the emulator is not connected. A resistor value of 4.7 kΩ or greater is suggested.

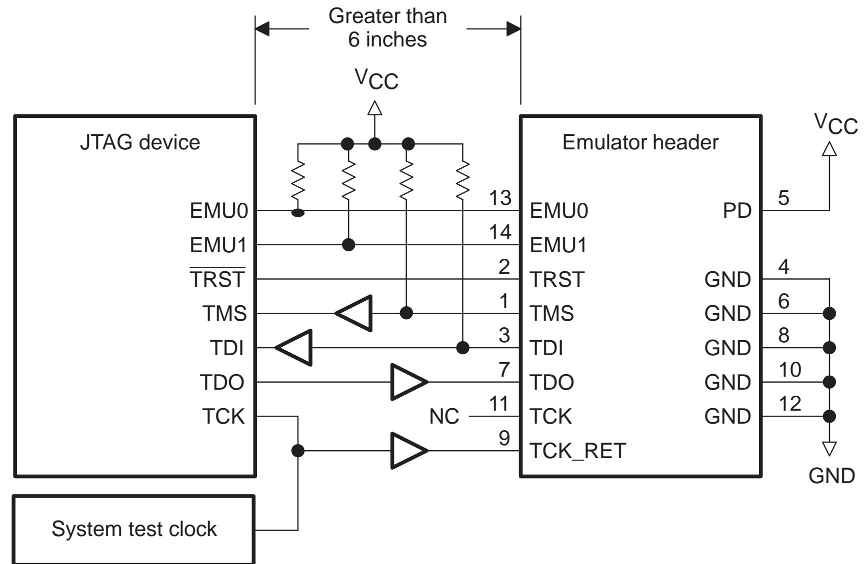
To have high-quality signals (especially the processor TCK and the emulator TCK_RET signals), you may have to employ special care when routing the printed wiring board trace. You also may have to use termination resistors to match the trace impedance. The emulator pod provides optional internal parallel terminators on the TCK_RET and TDO. TMS and TDI provide fixed series termination.

Because $\overline{\text{TRST}}$ is an asynchronous signal, it should be buffered as needed to ensure sufficient current to all target devices.

F.6.2 Using a Target-System Clock

Figure F–6 shows an application with the system test clock generated in the target system. In this application, the emulator's TCK signal is left unconnected.

Figure F–6. Target-System-Generated Test Clock



Note: When the TMS and TDI lines are buffered, pullup resistors must be used to hold the buffer inputs at a known level when the emulator cable is not connected.

There are two benefits in generating the test clock in the target system:

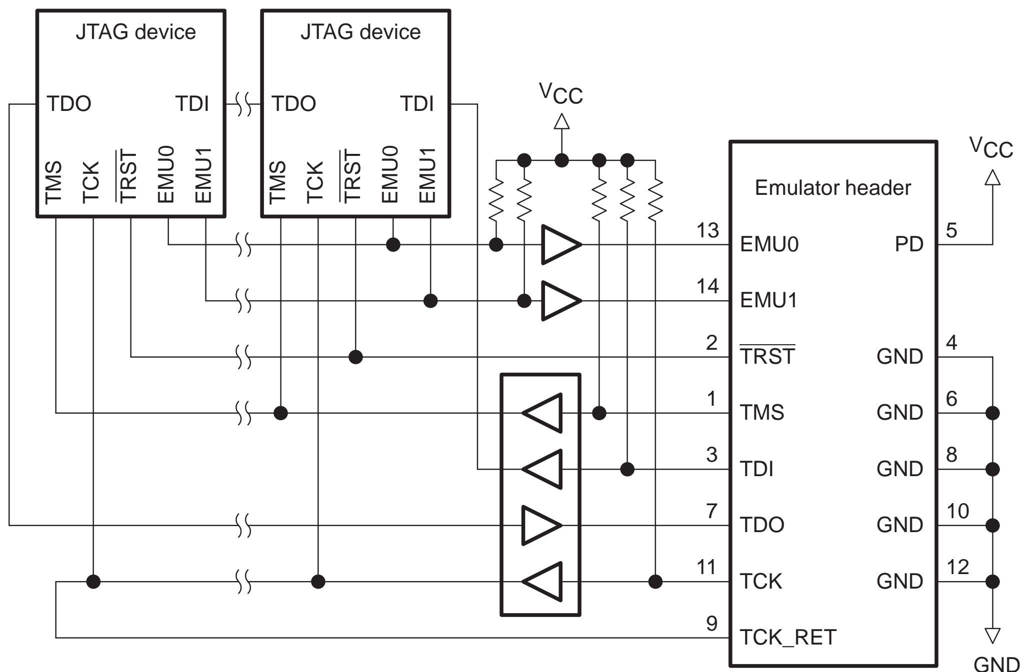
- The emulator provides only a single 10.368-MHz test clock. If you allow the target system to generate your test clock, you can set the frequency to match your system requirements.
- In some cases, you may have other devices in your system that require a test clock when the emulator is not connected. The system test clock also serves this purpose.

F.6.3 Configuring Multiple Processors

Figure F–7 shows a typical daisy-chained multiprocessor configuration that meets the minimum requirements of the IEEE 1149.1 specification. 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 interface is that you can slow down the test clock to eliminate timing problems. Follow these guidelines for multiprocessor support:

- The processor TMS, TDI, TDO, and TCK signals must 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 V_{CC} to hold these signals at a known value when the emulator is not connected. A 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.

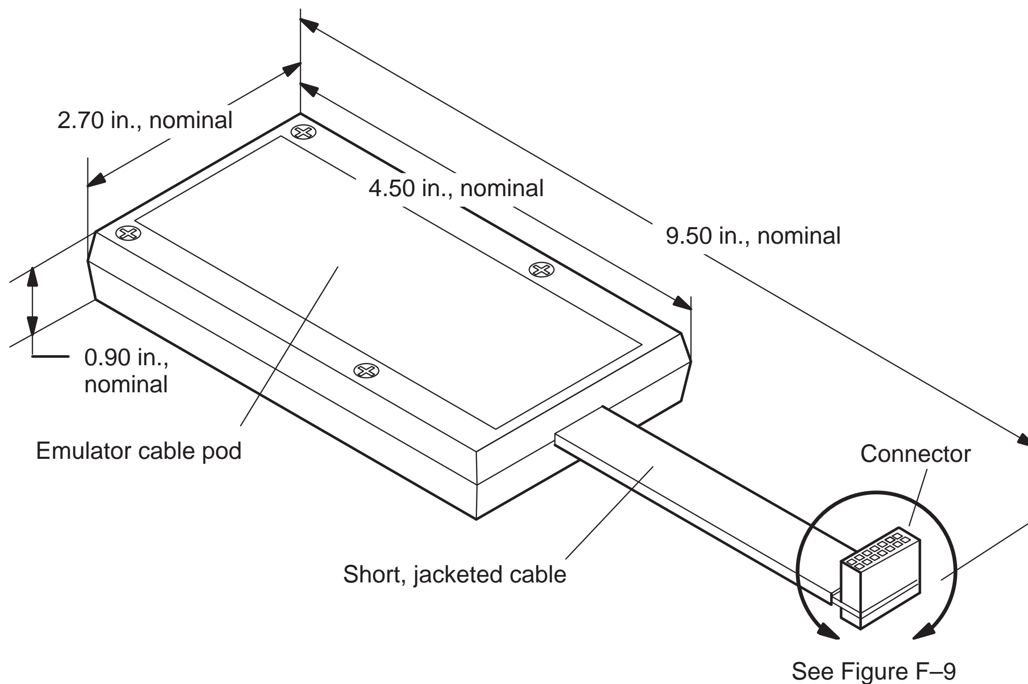
Figure F–7. Multiprocessor Connections



F.7 Physical Dimensions for the 14-Pin Emulator Connector

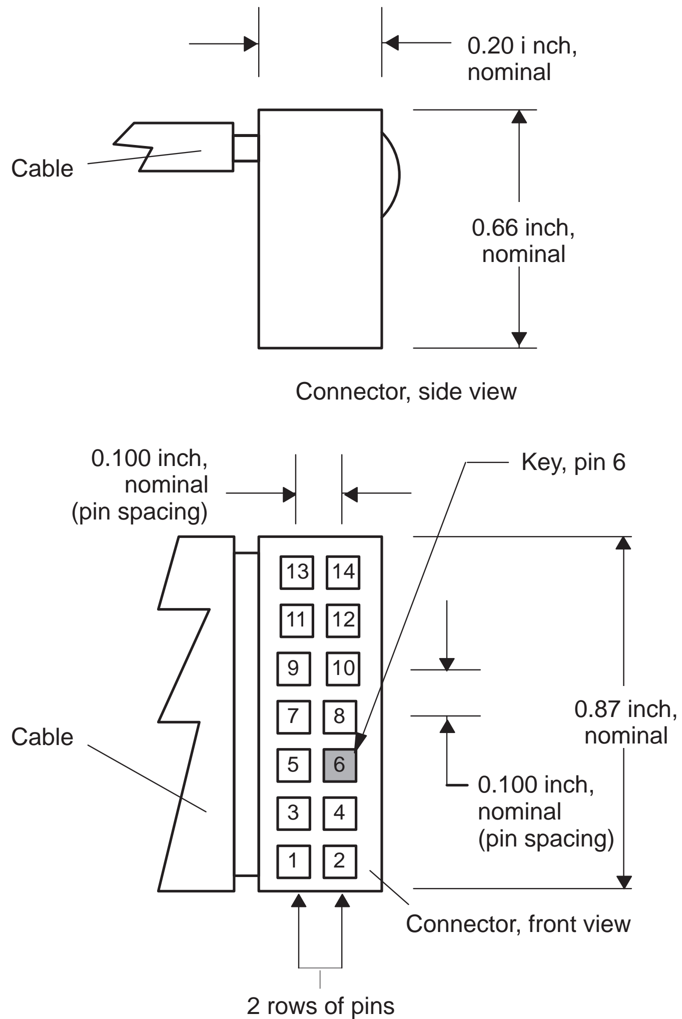
The JTAG emulator target cable consists of a 3-foot section of jacketed cable that connects to the emulator, an active cable pod, and a short section of jacketed cable that connects to the target system. The overall cable length is approximately 3 feet 10 inches. Figure F–8 and Figure F–9 (page F-15) show the physical dimensions for the target cable pod and short cable. The cable pod box is nonconductive plastic with four recessed metal screws.

Figure F–8. Pod/Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified. Pin-to-pin spacing on the connector is 0.100 inches in both the X and Y planes.

Figure F–9. 14-Pin Connector Dimensions



F.8 Emulation Design Considerations

This section describes the use and application of the scan path linker (SPL), which can simultaneously add all four secondary JTAG scan paths to the main scan path. It also describes the use of the emulation pins and the configuration of multiple processors.

F.8.1 Using Scan Path Linkers

You can use the TI ACT8997 scan path linker (SPL) to divide the JTAG emulation scan path into smaller, logically connected groups of 4 to 16 devices. As described in the *Advanced Logic and Bus Interface Logic Data Book*, the SPL is compatible with the JTAG emulation scanning. The SPL is capable of adding any combination of its four secondary scan paths into the main scan path.

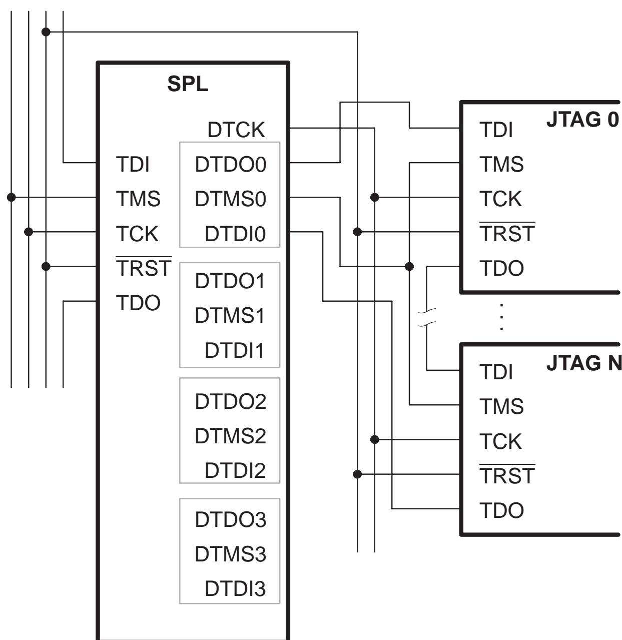
A system of multiple, secondary JTAG scan paths has better fault tolerance and isolation than a single scan path. Since an SPL has the capability of adding all secondary scan paths to the main scan path simultaneously, it can support global emulation operations, such as starting or stopping a selected group of processors.

TI emulators do not support the nesting of SPLs (for example, an SPL connected to the secondary scan path of another SPL). However, you can have multiple SPLs on the main scan path.

Scan path selectors are not supported by this emulation system. The TI ACT8999 scan path selector is similar to the SPL, but it can add only one of its secondary scan paths at a time to the main JTAG scan path. Thus, global emulation operations are not assured with the scan path selector.

You can insert an SPL on a backplane so that you can add up to four device boards to the system without the jumper wiring required with nonbackplane devices. You connect an SPL to the main JTAG scan path in the same way you connect any other device. Figure F–10 shows how to connect a secondary scan path to an SPL.

Figure F–10. Connecting a Secondary JTAG Scan Path to a Scan Path Linker



The TRST signal from the main scan path drives all devices, even those on the secondary scan paths of the SPL. The TCK signal on each target device on the secondary scan path of an SPL is driven by the SPL's DTCK signal. The TMS signal on each device on the secondary scan path is driven by the respective DTMS signals on the SPL.

DTDO0 on the SPL is connected to the TDI signal of the first device on the secondary scan path. DTDI0 on the SPL is connected to the TDO signal of the last device in the secondary scan path. Within each secondary scan path, the TDI signal of a device is connected to the TDO signal of the device before it. If the SPL is on a backplane, its secondary JTAG scan paths are on add-on boards; if signal degradation is a problem, you may need to buffer both the TRST and DTCK signals. Although degradation is less likely for DTMS*n* signals, you may also need to buffer them for the same reasons.

F.8.2 Emulation Timing Calculations for a Scan Path Linker (SPL)

Example F–3 and Example F–4 help you to calculate the key emulation timings in the SPL secondary scan path of your system. For actual target timing parameters, see the appropriate device data sheet for your target device.

The examples use the following assumptions:

$t_{su(TTMS)}$	Setup time, target TMS/TDI to TCK high	10 ns
$t_{d(TTDO)}$	Delay time, target TDO from TCK low	15 ns
$t_{d(bufmax)}$	Delay time, target buffer, maximum	10 ns
$t_{d(bufmin)}$	Delay time, target buffer, minimum	1 ns
$t_{(bufskew)}$	Skew time, target buffer, between two devices in the same package: $[t_{d(bufmax)} - t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{(TCKfactor)}$	Duty cycle, TCK assume a 40/60% clock	0.4 (40%)

Also, the examples use the following values from the SPL data sheet:

$t_{d(DTMSmax)}$	Delay time, SPL DTMS/DTDO from TCK low, maximum	31 ns
$t_{su(DTDLmin)}$	Setup time, DTDI to SPL TCK high, minimum	7 ns
$t_{d(DTCKHmin)}$	Delay time, SPL DTCK from TCK high, minimum	2 ns
$t_{d(DTCKLmax)}$	Delay time, SPL DTCK from TCK low, maximum	16 ns

There are two key timing paths to consider in the emulation design:

- The TCK-to-DTMS/DTDO path, called $t_{pd(TCK-DTMS)}$
- The TCK-to-DTDI path, called $t_{pd(TCK-DTDI)}$

Of the following two cases, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Example F–3. Key Timing for a Single-Processor System Without Buffering (SPL)

$$t_{pd\,(TCK\text{-}DTMS)} = \frac{\left[t_{d\,(DTMSmax)} + t_{d\,(DTCKHmin)} + t_{su\,(TTMS)}\right]}{t_{TCKfactor}}$$

$$= \frac{(31\text{ ns} + 2\text{ ns} + 10\text{ ns})}{0.4}$$

$$= 107.5\text{ ns, or } 9.3\text{ MHz}$$

$$t_{pd\,(TCK\text{-}DTDI)} = \frac{\left[t_{d\,(TTDO)} + t_{d\,(DTCKLmax)} + t_{su\,(DTDLmin)}\right]}{t_{TCKfactor}}$$

$$= \frac{(15\text{ ns} + 16\text{ ns} + 7\text{ ns})}{0.4}$$

$$= 9.5\text{ ns, or } 10.5\text{ MHz}$$

In this case, the TCK-to-DTMS/DTDL path is the limiting factor.

Example F–4. Key Timing for a Single- or Multiprocessor-System With Buffered Input and Output (SPL)

$$t_{pd\,(TCK\text{-}TDMS)} = \frac{\left[t_{d\,(DTMSmax)} + t_{(DTCKHmin)} + t_{su\,(TTMS)} + t_{(bufskew)}\right]}{t_{TCKfactor}}$$

$$= \frac{(31\text{ ns} + 2\text{ ns} + 10\text{ ns} + 1.35\text{ ns})}{0.4}$$

$$= 110.9\text{ ns, or } 9.0\text{ MHz}$$

$$t_{pd\,(TCK\text{–}DTDI)} = \frac{\left[t_{d\,(TTDO)} + t_{d\,(DTCKLmax)} + t_{su\,(DTDLmin)} + t_{d\,(bufskew)}\right]}{t_{TCKfactor}}$$

$$= \frac{(15\text{ ns} + 15\text{ ns} + 7\text{ ns} + 10\text{ ns})}{0.4}$$

$$= 120\text{ ns, or } 8.3\text{ MHz}$$

In this case, the TCK-to-DTDI path is the limiting factor.

F.8.3 Using Emulation Pins

The EMU0/1 pins of TI devices are bidirectional, 3-state output pins. When in an inactive state, these pins are at high impedance. When the pins are active, they provide one of two types of output:

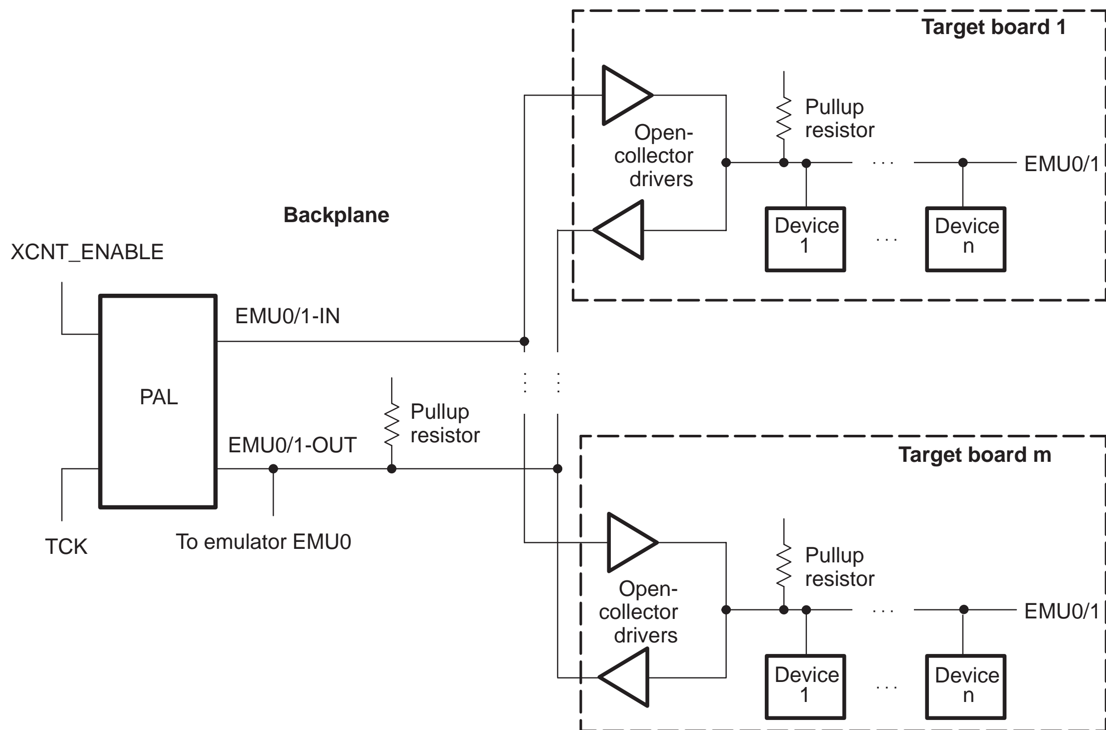
- Signal Event. The EMU0/1 pins can be configured via software to signal internal events. In this mode, driving one of these pins low can cause devices to signal such events. To enable this operation, the EMU0/1 pins function as open-collector sources. External devices such as logic analyzers can also be connected to the EMU0/1 signals in this manner. If such an external source is used, it must also be connected via an open-collector source.

- External Count. The EMU0/1 pins can be configured via software as totem-pole outputs for driving an external counter. If the output of more than one device is configured for totem-pole operation, then these devices can be damaged. The emulation software detects and prevents this condition. However, the emulation software has no control over external sources on the EMU0/1 signal. Therefore, all external sources must be inactive when any device is in the external count mode.

TI devices can be configured by software to halt processing if their EMU0/1 pins are driven low. This feature combined with the signal event output, allows one TI device to halt all other TI devices on a given event for system-level debugging.

If you route the EMU0/1 signals between multiple boards, they require special handling because they are more complex than normal emulation signals. Figure F–11 shows an example configuration that allows any processor in the system to stop any other processor in the system. Do not tie the EMU0/1 pins of more than 16 processors together in a single group without using buffers. Buffers provide the crisp signals that are required during a RUNB (run benchmark) debugger command or when the external analysis counter feature is used.

Figure F–11.EMU0/1 Configuration to Meet Timing Requirements of Less Than 25 ns



Notes:
1) The low time on EMU0/1-IN should be at least one TCK cycle and less than 10 μs. Software sets the EMU0/1-OUT pin to a high state.
2) To enable the open-collector driver and pullup resistor on EMU1 to provide rise/fall times of less than 25 ns, the modification shown in this figure is suggested. Rise times of more than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used.

These seven important points apply to the circuitry shown in Figure F–11 and the timing shown in Figure F–12:

- Open-collector drivers isolate each board. The EMU0/1 pins are tied together on each board.
- At the board edge, the EMU0/1 signals are split to provide both input and output connections. This is required to prevent the open-collector drivers from acting as latches that can be set only once.
- The EMU0/1 signals are bused down the backplane. Pullup resistors must be installed as required.

- The bused EMU0/1 signals go into a programmable logic array device PAL® whose function is to generate a low pulse on the EMU0/1-IN signal when a low level is detected on the EMU0/1-OUT signal. This pulse must be longer than one TCK period to affect the devices but less than 10 μs to avoid possible conflicts or retriggering once the emulation software clears the device's pins.

- During a RUNB debugger command or other external analysis count, the EMU0/1 pins on the target device become totem-pole outputs. The EMU1 pin is a ripple carry-out of the internal counter. EMU0 becomes a *processor-halted* signal. During a RUNB or other external analysis count, the EMU0/1-IN signal to all boards must remain in the high (disabled) state. You must provide some type of external input (XCNT_ENABLE) to the PAL® to disable the PAL® from driving EMU0/1-IN to a low state.

- If you use sources other than TI processors (such as logic analyzers) to drive EMU0/1, their signal lines must be isolated by open-collector drivers and be inactive during RUNB and other external analysis counts.

- You must connect the EMU0/1-OUT signals to the emulation header or directly to a test bus controller.

Figure F–12. Suggested Timings for the EMU0 and EMU1 Signals

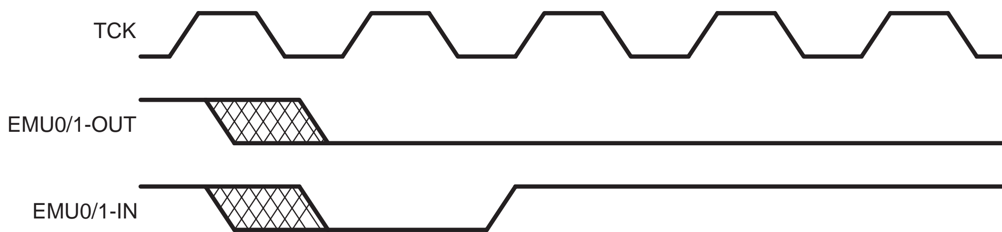
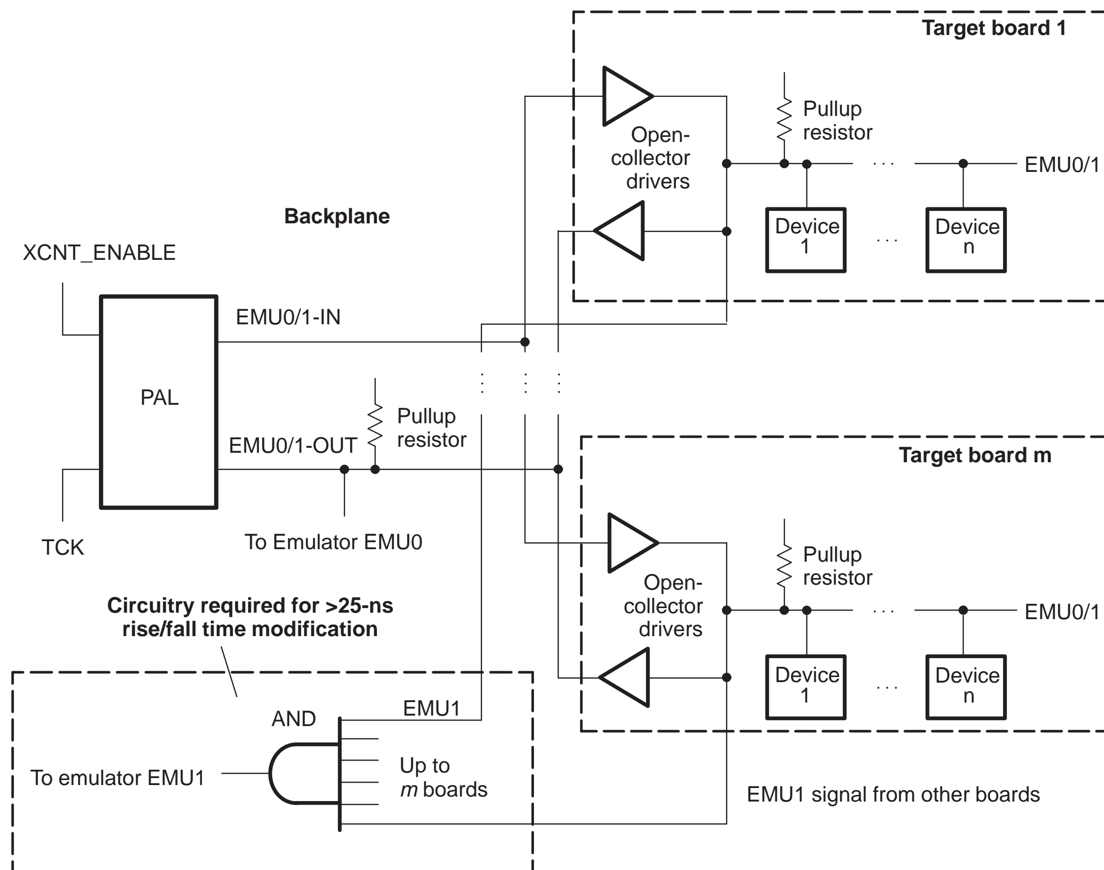


Figure F–13. EMU0/1 Configuration With Additional AND Gate to Meet Timing Requirements of Greater Than 25 ns

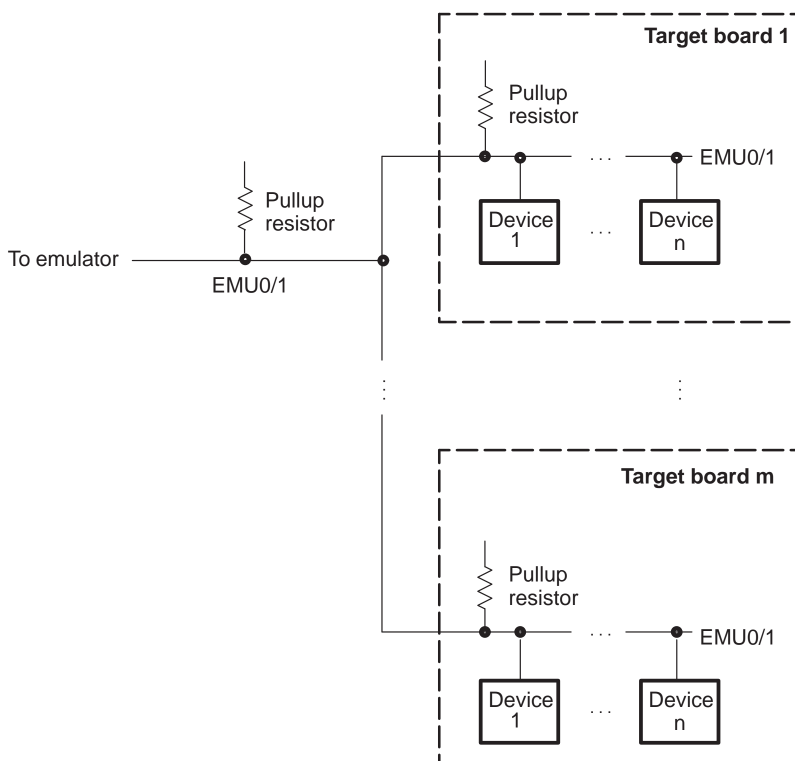


Notes: 1) The low time on EMU0/1-IN should be at least one TCK cycle and less than 10 μs. Software will set the EMU0/1-OUT port to a high state.

2) To enable the open-collector driver and pullup resistor on EMU1 to provide rise/fall time of greater than 25 ns, the modification shown in this figure is suggested. Rise times of more than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used.

You do not need to have devices on one target board stop devices on another target board using the EMU0/1 signals (see the circuit in Figure F–14). In this configuration, the global-stop capability is lost. It is important not to overload EMU0/1 with more than 16 devices.

Figure F–14. EMU0/1 Configuration Without Global Stop

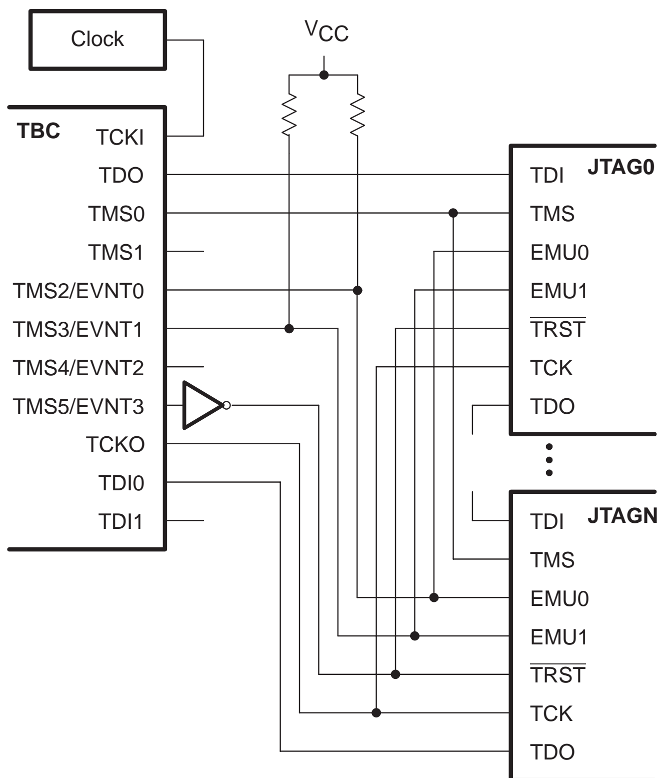


Note: The open-collector driver and pullup resistor on EMU1 must be able to provide rise/fall times of less than 25 ns. Rise times of more than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used. If this condition cannot be met, then the EMU0/1 signals from the individual boards must be ANDed together (as shown in Figure F–14) to produce an EMU0/1 signal for the emulator.

F.8.4 Performing Diagnostic Applications

For systems that require built-in diagnostics, it is possible to connect the emulation scan path directly to a TI ACT8990 test bus controller (TBC) instead of the emulation header. The TBC is described in the Texas Instruments *Advanced Logic and Bus Interface Logic Data Book*. Figure F–15 shows the scan path connections of *n* devices to the TBC.

Figure F–15. TBC Emulation Connections for n JTAG Scan Paths



In the system design shown in Figure F–15, the TBC emulation signals TCKI, TDO, TMS0, TMS2/EVNT0, TMS3/EVNT1, TMS5/EVNT3, TCKO, and TDI0 are used, and TMS1, TMS4/EVNT2, and TDI1 are not connected. The target devices' EMU0 and EMU1 signals are connected to V_{CC} through pullup resistors and tied to the TBC's TMS2/EVNT0 and TMS3/EVNT1 pins, respectively. The TBC's TCKI pin is connected to a clock generator. The TCK signal for the main JTAG scan path is driven by the TBC's TCKO pin.

On the TBC, the TMS0 pin drives the TMS pins on each device on the main JTAG scan path. TDO on the TBC connects to TDI on the first device on the main JTAG scan path. TDI0 on the TBC is connected to the TDO signal of the last device on the main JTAG scan path. Within the main JTAG scan path, the TDI signal of a device is connected to the TDO signal of the device before it. $\overline{TRST}$ for the devices can be generated either by inverting the TBC's TMS5/EVNT3 signal for software control or by logic on the board itself.

Appendix G

Glossary

A

A0–A15: Collectively, the external address bus; the 16 pins are used in parallel to address external data memory, program memory, or I/O space.

ACC: See *accumulator.*

ACCH: *Accumulator high word.* The upper 16 bits of the accumulator. See also *accumulator*.

ACCL: *Accumulator low word.* The lower 16 bits of the accumulator. See also *accumulator*.

accumulator: A 32-bit register that stores the results of operations in the central arithmetic logic unit (CALU) and provides an input for subsequent CALU operations. The accumulator also performs shift and rotate operations.

ADC bit: A *detect complete bit*. Bit 14 of the I/O status register (IOSR); a flag bit used in the implementation of automatic baud-rate detection in the asynchronous serial port.

address: The location of program code or data stored in memory.

addressing mode: A method by which an instruction interprets its operands to acquire the data it needs. See also *direct addressing*; *immediate addressing*; *indirect addressing*.

address visibility bit (AVIS): A bit in the ’C209’s wait-state generator control register (WSGR) that allows the internal program address to appear at the ’C209 address pins. This allows the internal program address to be traced.

ADTR: *Asynchronous data transmit and receive register.* A 16-bit register used by the on-chip asynchronous serial port. Data to transmit is written to the 8 LSBs of the ADTR, and received data is read from the 8 LSBs of the ADTR. See also *ARSR.*

analog-to-digital (A/D) converter: A circuit that translates an analog signal to a digital signal.

AR: See *auxiliary register*.

AR0–AR7: *Auxiliary registers 0 through 7.* See *auxiliary register*.

ARAU: See *auxiliary register arithmetic unit (ARAU).*

ARB: See *auxiliary register pointer buffer (ARB)*.

ARP: See *auxiliary register pointer (ARP).*

ARSR: *Asynchronous serial port receive shift register.* A 16-bit register in the on-chip asynchronous serial port that receives data from the RX pin one bit at a time. When full, ARSR transfers its data to the ADTR. See also *ADTR.*

ASPCR: *Asynchronous serial port control register*. A 16-bit register used to control the on-chip asynchronous serial port; contains bits for setting port modes, enabling or disabling the automatic baud-rate detection logic, selecting the number of stop bits, enabling or disabling interrupts, setting the default level on the TX pin, configuring pins IO3–IO0, and resetting the port.

auxiliary register: One of eight 16-bit registers (AR7–AR0) used as pointers to addresses in data space. The registers are operated on by the auxiliary register arithmetic unit (ARAU) and are selected by the auxiliary register pointer (ARP).

auxiliary register arithmetic unit (ARAU): A 16-bit arithmetic unit used to increment, decrement, or compare the contents of the auxiliary registers. Its primary function is manipulating auxiliary register values for indirect addressing.

auxiliary register pointer (ARP): A 3-bit field in status register ST0 that points to the current auxiliary register.

auxiliary register pointer buffer (ARB): A 3-bit field in status register ST1 that holds the previous value of the auxiliary register pointer (ARP).

AVIS: See *address visibility bit (AVIS)*.

AXSR: *Asynchronous serial port transmit shift register.* A 16-bit register in the asynchronous serial port that receives data from the ADTR and transfers it one bit at a time to the TX pin. See also *ADTR*; *TX pin.*

B

B0: An on-chip block of dual-access RAM that can be configured as either data memory or program memory, depending on the value of the CNF bit in status register ST1.

B1: An on-chip block of dual-access RAM available for data memory.

B2: An on-chip block of dual-access RAM available for data memory.

baud-rate divisor register (BRD): A register for the asynchronous serial port that is used to set the serial port's baud rate.

BI bit: *Break interrupt bit*. Bit 13 of the I/O status register (IOSR); indicates when a break is detected on the asynchronous receive (RX) pin.

$\overline{\text{BIO}}$ pin: A general-purpose input pin that can be tested by conditional instructions that cause a branch when an external device drives $\overline{\text{BIO}}$ low.

bit-reversed indexed 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.

bootloader: A built-in segment of code that transfers code from an 8-bit external source to a 16-bit external program destination at reset.

$\overline{\text{BOOT}}$ pin: The pin that enables the on-chip bootloader. When $\overline{\text{BOOT}}$ is held low, the processor executes the bootloader program after a hardware reset. When $\overline{\text{BOOT}}$ is held high, the processor skips execution of the bootloader and accesses off-chip program-memory at reset.

$\overline{\text{BR}}$: *Bus request pin*. This pin is tied to the $\overline{\text{BR}}$ signal, which is asserted when a global data memory access is initiated.

branch: A switching of program control to a nonsequential program-memory address.

BRD: See *baud-rate divisor register (BRD).*

burst mode: A synchronous serial port mode in which the transmission or reception of each word is preceded by a frame synchronization pulse. See also *continuous mode*.

C

C bit: See *carry bit (C).*

CAD bit: *Calibrate* A *detect bit*. Bit 5 of the ASPCR; enables and disables the automatic baud-rate detection logic of the on-chip asynchronous serial port.

CALU: See *central arithmetic logic unit (CALU).*

carry bit: Bit 9 of status register ST1; used by the CALU for extended arithmetic operations and accumulator shifts and rotates. The carry bit can be tested by conditional instructions.

central arithmetic logic unit (CALU): The 32-bit wide main arithmetic logic unit for the ’C2xx CPU that performs arithmetic and logic operations. It accepts 32-bit values for operations, and its 32-bit output is held in the accumulator.

CIO0–CIO3 bits: Bits 0–3 of the asynchronous serial port control register (ASPCR); they individually configure pins IO0–IO3 as either inputs or outputs. For example, CIO0 configures the IO0 pin. See also *DIO0–DIO3 bits*; *IO0–IO3 bits*.

CLK register: *CLKOUT1-pin control register.* Bit 0 of determines whether the CLKOUT1 signal is available at the CLKOUT1 pin.

CLKIN: *Input clock signal*. A clock source signal supplied to the on-chip clock generator at the CLKIN/X2 pin or generated internally by the on-chip oscillator. The clock generator divides or multiplies CLKIN to produce the CPU clock signal, CLKOUT1.

CLKMOD pin: (On the ’C209 only) Determines whether the on-chip clock generator is running in the divide-by-two or multiply-by-two mode. See also *clock mode*.

CLKOUT1: *Master clock output signal*. The output signal of the on-chip clock generator. The CLKOUT1 high pulse signifies the CPU’s logic phase (when internal values are changed), and the CLKOUT1 low pulse signifies the CPU’s latch phase (when the values are held constant).

CLKOUT1 cycle: See *CPU cycle*.

CLKOUT1-pin control register: See *CLK register*.

CLKR: *Receive clock input pin.* A pin that receives an external clock signal to clock data from the DR pin into the synchronous serial port receive shift register (RSR).

CLKX: *Transmit clock input/output pin.* A pin used to clock data from the synchronous serial port transmit shift register to the DX pin. If the serial port is configured to accept an external clock, this pin receives the clock signal. If the port is configured to generate an internal clock, this pin transmits the clock signal.

clock mode (clock generator): One of the modes which sets the internal CPU clock frequency to a fraction or multiple of the frequency of the input clock signal CLKIN. The ’C209 has two clock modes (÷2 and ×2); other ’C2xx devices have four clock modes (÷2, ×1, ×2, and ×4).

clock mode (synchronous serial port): See *clock mode bit (MCM).*

clock mode bit (MCM): Bit 2 of the synchronous serial port control register (SSPCR); determines whether the source signal for clocking synchronous serial port transfers is external or internal.

CNF bit: *DARAM configuration bit*. Bit 12 in status register ST1. CNF is used to determine whether the on-chip RAM block B0 is mapped to program space or data space.

codec: A device that codes in one direction of transmission and decodes in another direction of transmission.

COFF: *Common object file format.* An output format that promotes modular programming by supporting sections; the format of files created by the TMS320C1x/C2x/C2xx/C5x assembler and linker.

context saving/restoring: Saving the system status when the device enters a subroutine (such as an interrupt service routine) and restoring the system status when exiting the subroutine. On the ’C2xx, only the program counter value is saved and restored automatically; other context saving and restoring must be performed by the subroutine.

continuous mode: A synchronous serial port mode in which only one frame synchronization pulse is necessary to transmit or receive several consecutive packets at maximum frequency. See also *burst mode*.

CPU: *Central processing unit*. The ’C2xx CPU is the portion of the processor involved in arithmetic, shifting, and Boolean logic operations, as well as the generation of data- and program-memory addresses. The CPU includes the central arithmetic logic unit (CALU), the multiplier, and the auxiliary register arithmetic unit (ARAU).

CPU cycle: The time required for the CPU to go through one logic phase (during which internal values are changed) and one latch phase (during which the values are held constant).

current AR: See *current auxiliary register*.

current auxiliary register: The auxiliary register pointed to by the auxiliary register pointer (ARP). The auxiliary registers are AR0 (ARP = 0) through AR7 (ARP = 7). See also *auxiliary register*, *next auxiliary register*.

current data page: The data page indicated by the content of the data page pointer (DP). See also *data page*; *DP*.

D

D0–D15: Collectively, the external data bus; the 16 pins are used in parallel to transfer data between the ’C2xx and external data memory, program memory, or I/O space.

DARAM: *Dual-access RAM*. RAM that can be accessed twice in a single CPU clock cycle. For example, your code can read from and write to DARAM in the same clock cycle.

DARAM configuration bit (CNF): See *CNF bit*.

data-address generation logic: Logic circuitry that generates the addresses for data memory reads and writes. This circuitry, which includes the auxiliary registers and the ARAU, can generate one address per machine cycle. See also *program-address generation logic.*

data page: One block of 128 words in data memory. Data memory contains 512 data pages. Data page 0 is the first page of data memory (addresses 0000h–007Fh); data page 511 is the last page (addresses FF80h–FFFFh). See also *data page pointer (DP)*; *direct addressing*.

data page 0: Addresses 0000h–007Fh in data memory; contains the memory-mapped registers, a reserved test/emulation area for special information transfers, and the scratch-pad RAM block (B2).

data page pointer (DP): A 9-bit field in status register ST0 that specifies which of the 512 data pages is currently selected for direct address generation. When an instruction uses direct addressing to access a data-memory value, the DP provides the nine MSBs of the data-memory address, and the instruction provides the seven LSBs.

data-read address bus (DRAB): A 16-bit internal bus that carries the address for each read from data memory.

data read bus (DRDB): A 16-bit internal bus that carries data from data memory to the CALU and the ARAU.

data-write address bus (DWAB): A 16-bit internal bus that carries the address for each write to data memory.

data write bus (DWEB): A 16-bit internal bus that carries data to both program memory and data memory.

decode phase: The phase of the pipeline in which the instruction is decoded. See also *pipeline*; *instruction-fetch phase*; *operand-fetch phase; instruction-execute phase*.

delta interrupt: An asynchronous serial port interrupt (TXRXINT) that is generated if a change takes place on one of these general-purpose I/O pins: IO0, IO1, IO2, or IO3.

digital loopback mode: A synchronous serial port test mode in which the receive pins are connected internally to the transmit pins on the same device. This mode, enabled or disabled by the DLB bit, allows you to test whether the port is operating correctly.

DIM: *Delta interrupt mask bit*. Bit 9 of the asynchronous serial port control register (ASPCR); enables or disables delta interrupts.

DIO0–DIO3 bits: Bits 4–7 of the IOSR. If the asynchronous serial port is enabled (the URST bit of the ASPCR is 1), these bits are used to track a change from a previous known or unknown signal value at the corresponding I/O pin (IO0–IO3). For example, DIO0 indicates a change on the IO0 pin. See also *CIO0–CIO3 bits*; *IO0–IO3 bits*.

direct addressing: One of the methods used by an instruction to address data-memory. In direct addressing, the data-page pointer (DP) holds the nine MSBs of the address (the current data page), and the instruction word provides the seven LSBs of the address (the offset). See also *indirect addressing*.

DIV2/DIV1: Two pins used together to determine the clock mode of the 'C2xx clock generator (÷2, ×1, ×2, or ×4). (The 'C209 uses the CLKMOD pin and has only two clock modes, ÷2 and ×2.)

divide-down value: The value in the timer divide-down register (TDDR). This value is the prescale count for the on-chip timer. The larger the divide-down value, the slower the timer interrupt rate.

DLB bit: Bit 0 of the synchronous serial port control register (SSPCR); enables or disables digital loopback mode for the on-chip synchronous serial port. See also *digital loopback mode*.

DP: See *data page pointer (DP).*

DR bit: *Data ready indicator for the receiver.* Bit 8 of the I/O status register (IOSR); indicates whether a new 8-bit character has been received in the ADTR of the asynchronous serial port.

DR pin: *Serial data receive pin.* A synchronous serial port pin that receives serial data. As each bit is received at DR, the bit is transferred serially into the receive shift register (RSR).

DRAB: See *data-read address bus (DRAB).*

DRDB: See *data read bus (DRDB).*

$\overline{\text{DS}}$: *Data memory select pin*. The 'C2xx asserts $\overline{\text{DS}}$ to indicate an access to external data memory (local or global).

DSWS: *Data-space wait-state bit(s).* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip data space. On the 'C209, DSWS is bit 1 of the WSGR; on other 'C2xx devices, DSWS is bits 8–6.

dual-access RAM: See *DARAM.*

dummy cycle: A CPU cycle in which the CPU intentionally reloads the program counter with the same address.

DWAB: See *data-write address bus (DWAB).*

DWEB: See *data write bus (DWEB).*

DX pin: *Serial data transmit pin*. The pin on which data is transmitted serially from the synchronous serial port; accepts a data word one bit at a time from the transmit shift register (XSR).

E

execute phase: The fourth phase of the pipeline; the phase in which the instruction is executed. See also *pipeline*; *instruction-fetch phase*; *instruction-decode phase*; *operand-fetch phase*.

external interrupt: A hardware interrupt triggered by an external event sending an input through an interrupt pin.

F

FE bit: *Framing error indicator bit*. Bit 10 of I/O status register (IOSR); indicates whether a valid stop bit has been detected during the reception of a character into the asynchronous serial port.

FIFO buffer: *First-in, first-out buffer.* A portion of memory in which data is stored and then retrieved in the same order in which it was stored. The synchronous serial port has two four-word-deep FIFO buffers: one for its transmit operation and one for its receive operation.

flash memory: Electronically erasable and programmable, nonvolatile (read-only) memory.

FR0/FR1: *FIFO receive-interrupt bits*. Bits 8 and 9 of the synchronous serial port control register (SSPCR); together they set an interrupt trigger condition based on the number of words in the receive FIFO buffer.

frame synchronization (frame sync) mode: One of two modes in the synchronous serial port that determine whether frame synchronization pulses are necessary between consecutive data transfers. See also *burst mode*; *continuous mode*.

frame synchronization (frame sync) pulse: A pulse that signals the start of a transmission from or reception into the synchronous serial port.

framing error: An error that occurs when a data character received by the asynchronous serial port does not have a valid stop bit. See also *FE bit*.

FREE bit (asynchronous serial port): Bit 15 of the asynchronous serial port control register (ASPCR); determines whether the port is in free-run mode or an emulation mode. When FREE = 0, bit 14 (SOFT) determines which emulation mode is selected.

FREE bit (synchronous serial port): Bit 15 of the synchronous serial port control register (SSPCR); determines whether the port is in free-run mode or an emulation mode. When FREE = 0, bit 14 (SOFT) determines which emulation mode is selected.

FREE bit (timer): Bit 11 of the timer control register (TCR); determines whether the timer is in free-run mode or an emulation mode. When FREE = 0, bit 14 (SOFT) determines which emulation mode is selected. FREE and SOFT are not available in the TCR of the ’C209.

FSM bit: Bit 1 of the synchronous serial port control register (SSPCR); determines the frame synchronization mode for the synchronous serial port. See also *burst mode*; *continuous mode*.

FSR pin: *Receive frame synchronization pin.* This input pin accepts a frame sync pulse that initiates the reception process of the synchronous serial port.

FSX pin: *Transmit frame synchronization pin.* This input/output pin accepts/generates a frame sync pulse that initiates the transmission process of the synchronous serial port. If the port is configured for accepting an external frame sync pulse, the FSX pin receives the pulse. If the port is configured for generating an internal frame sync pulse, the FSX pin transmits the pulse.

FT0/FT1: *FIFO transmit-interrupt bits.* Bits 10 and 11 of the synchronous serial port control register (SSPCR); together they set an interrupt trigger condition based on the number of words in the transmit FIFO buffer.

G

general-purpose input/output pins: Pins that can be used to accept input signals and/or send output signals but are not linked to specific uses. These pins are the input pin $\overline{\text{BIO}}$, the output pin XF, and the input/output pins IO0, IO1, IO2, and IO3. (IO0–IO3 are not available on the ’C209.)

global data space: One of the four ’C2xx address spaces. The global data space can be used to share data with other processors within a system and can serve as additional data space. See also *local data space*.

GREG: *Global memory allocation register.* A memory-mapped register used for specifying the size of the global data memory. Addresses not allocated by the GREG for global data memory are available for local data memory.

H

hardware interrupt: An interrupt triggered through physical connections with on-chip peripherals or external devices.

$\overline{\text{HOLD}}$: An input signal that allows external devices to request control of the external buses. If an external device drives the $\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin low and the CPU sends an acknowledgement at the $\overline{\text{HOLDA}}$ pin, the external device has control of the buses until it drives $\overline{\text{HOLD}}$ high or a nonmaskable hardware interrupt is generated. If $\overline{\text{HOLD}}$ is not used, it should be pulled high.

$\overline{\text{HOLDA}}$: *$\overline{\text{HOLD}}$ acknowledge signal*. An output signal sent to the $\overline{\text{HOLDA}}$ pin by the CPU in acknowledgement of a properly initiated HOLD operation. When $\overline{\text{HOLDA}}$ is low, the processor is in a holding state and the address, data, and memory-control lines are available to external circuitry.

HOLD operation: An operation on the ’C2xx that allows for direct memory access of external memory and I/O devices. A HOLD operation is initiated by a $\overline{\text{HOLD}}/\overline{\text{INT1}}$ interrupt. When the corresponding interrupt service routine executes an IDLE instruction, the external buses enter the high-impedance state and the $\overline{\text{HOLDA}}$ signal is asserted. The buses return to their normal state, and the HOLD operation is concluded, when the processor exits the IDLE state.

I

$\overline{\text{IACK}}$: See *interrupt acknowledge signal ($\overline{\text{IACK}}$)*.

IC: (Used in earlier documentation.) See *interrupt control register (ICR).*

ICR: See *interrupt control register (ICR)*.

IFR: See *interrupt flag register (IFR).*

immediate addressing: One of the methods for obtaining data values used by an instruction; the data value is a constant embedded directly into the instruction word; data memory is not accessed.

immediate operand/immediate value: A constant given as an operand in an instruction that is using immediate addressing.

IMR: See *interrupt mask register (IMR)*.

IN0: Bit 6 of the synchronous serial port control register (SSPCR); allows you to use the CLKR pin as a bit input. IN0 indicates the current logic level on CLKR.

indirect addressing: One of the methods for obtaining data values used by an instruction. When an instruction uses indirect addressing, data memory is addressed by the current auxiliary register. See also *direct addressing*.

input clock signal: See *CLKIN*.

input/output status register: See I/O status register (IOSR).

input shifter: A 16- to 32-bit left barrel shifter that shifts incoming 16-bit data from 0 to 16 positions left relative to the 32-bit output.

instruction-decode phase: The second phase of the pipeline; the phase in which the instruction is decoded. See also *pipeline*; *instruction-fetch phase*; *operand-fetch phase; instruction-execute phase*.

instruction-execute phase: The fourth phase of the pipeline; the phase in which the instruction is executed. See also *pipeline*; *instruction-fetch phase*; *instruction-decode phase*; *operand-fetch phase*.

instruction-fetch phase: The first phase of the pipeline; the phase in which the instruction is fetched from program-memory. See also *pipeline*; *instruction-decode phase*; *operand-fetch phase; instruction-execute phase*.

instruction register (IR): A 16-bit register that contains the instruction being executed.

instruction word: A 16-bit value representing all or half of an instruction. An instruction that is fully represented by 16 bits uses one instruction word. An instruction that must be represented by 32 bits uses two instruction words (the second word is a constant).

$\overline{\text{INT1}}$–$\overline{\text{INT3}}$: Three external pins used to generate general-purpose hardware interrupts.

internal interrupt: A hardware interrupt caused by an on-chip peripheral.

interrupt: A signal sent to the CPU that (when not masked or disabled) forces the CPU into a subroutine called an interrupt service routine (ISR). This signal can be triggered by an external device, an on-chip peripheral, or an instruction (INTR, NMI, or TRAP).

interrupt acknowledge signal ($\overline{\text{IACK}}$): An output signal on the ’C209 that indicates that an interrupt has been received and that the program counter is fetching the interrupt vector that will force the processor into the appropriate interrupt service routine.

interrupt control register (ICR): A 16-bit register used to differentiate $\overline{\text{HOLD}}$ and $\overline{\text{INT1}}$ and to individually mask and flag $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$.

interrupt flag register (IFR): A 16-bit memory-mapped register that indicates pending interrupts. Read the IFR to identify pending interrupts and write to the IFR to clear selected interrupts. Writing a 1 to any IFR flag bit clears that bit to 0.

interrupt latency: The delay between the time an interrupt request is made and the time it is serviced.

interrupt mask register (IMR): A 16-bit memory-mapped register used to mask external and internal interrupts. Writing a 1 to any IMR bit position enables the corresponding interrupt (when INTM = 0).

interrupt mode bit (INTM): Bit 9 in status register ST0; either enables all maskable interrupts that are not masked by the IMR or disables all maskable interrupts.

interrupt service routine (ISR): A module of code that is executed in response to a hardware or software interrupt.

interrupt trap: See *interrupt service routine (ISR)*.

interrupt vector: A branch instruction that leads the CPU to an interrupt service routine (ISR).

interrupt vector location: An address in program memory where an interrupt vector resides. When an interrupt is acknowledged, the CPU branches to the interrupt vector location and fetches the interrupt vector.

INTM bit: See *interrupt mode bit (INTM)*.

IO0–IO3 bits: Bits 0–3 of the IOSR. When pins IO0–IO3 are configured as inputs, these bits reflect the current logic levels on the pins. For example, the IO0 bit reflects the level on the IO0 pin. See also *CIO0–CIO3 bits*; *DIO0–DIO3 bits*.

IO0–IO3 pins: Four pins that can be individually configured as inputs or outputs. These pins can be used for interfacing the asynchronous serial port or as general-purpose I/O pins. See also *CIO0–CIO3 bits; DIO0–DIO3 bits; IO0–IO3 bits*.

I/O-mapped register: One of the on-chip registers mapped to addresses in I/O (input/output) space. These registers, which include the registers for the on-chip peripherals, must be accessed with the IN and OUT instructions. See also *memory-mapped register*.

I/O status register (IOSR): A register in the asynchronous serial port that provides status information about signals IO0–IO3 and about transfers in progress.

IOSR: See *I/O status register (IOSR).*

IR: See *instruction register (IR).*

$\overline{\text{IS}}$: *I/O space select pin*. The ’C2xx asserts $\overline{\text{IS}}$ to indicate an access to external I/O space.

ISR: See *interrupt service routine (ISR)*.

ISWS: *I/O-space wait-state bit(s).* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip I/O space. On the ’C209, ISWS is bit 2 of the WSGR; on other ’C2xx devices, ISWS is bits 11–9.

L

latch phase: The phase of a CPU cycle during which internal values are held constant. See also *logic phase*; *CLKOUT1*.

local data space: The portion of data-memory addresses that are not allocated as global by the global memory allocation register (GREG). If none of the data-memory addresses are allocated for global use, all of data space is local. See also *global data space*.

logic phase: The phase of a CPU cycle during which internal values are changed. See also *latch phase*; *CLKOUT1*.

long-immediate value: A 16-bit constant given as an operand of an instruction that is using immediate addressing.

LSB: *Least significant bit.* The lowest order bit in a word. When used in plural form (LSBs), refers to a specified number of low-order bits, beginning with the lowest order bit and counting to the left. For example, the four LSBs of a 16-bit value are bits 0 through 3. See also *MSB*.

M

machine cycle: See *CPU cycle*.

maskable interrupt: A hardware interrupt that can be enabled or disabled through software. See also *nonmaskable interrupt*.

master clock output signal: See *CLKOUT1*.

master phase: See *logic phase*.

MCM bit: See *clock mode bit (MCM)*.

memory-mapped register: One of the on-chip registers mapped to addresses in data memory. See also *I/O-mapped register*.

microcomputer mode: A mode in which the on-chip ROM or flash memory is enabled. This mode is selected with the MP/$\overline{\text{MC}}$ pin. See also *MP/$\overline{\text{MC}}$ pin*; *microprocessor mode*.

microprocessor mode: A mode in which the on-chip ROM or flash memory is disabled. This mode is selected with the MP/$\overline{\text{MC}}$ pin. See also *MP/$\overline{\text{MC}}$ pin*; *microcomputer mode*.

micro stack (MSTACK): A register used for temporary storage of the program counter (PC) value when an instruction needs to use the PC to address a second operand.

MIPS: Million instructions per second.

MODE bit: Bit 4 of the interrupt control register (ICR); determines whether the $\overline{\text{HOLD}}$/$\overline{\text{INT1}}$ pin is only negative-edge sensitive or both negative- and positive-edge sensitive.

MP/$\overline{\text{MC}}$ pin: A pin that indicates whether the processor is operating in microprocessor mode or microcomputer mode. MP/$\overline{\text{MC}}$ high selects microprocessor mode; MP/$\overline{\text{MC}}$ low selects microcomputer mode.

MSB: *Most significant bit.* The highest order bit in a word. When used in plural form (MSBs), refers to a specified number of high-order bits, beginning with the highest order bit and counting to the right. For example, the eight MSBs of a 16-bit value are bits 15 through 8. See also *LSB*.

MSTACK: See *micro stack*.

multiplier: A part of the CPU that performs 16-bit × 16-bit multiplication and generates a 32-bit product. The multiplier operates using either signed or unsigned 2s-complement arithmetic.

N

next AR: See *next auxiliary register*.

next auxiliary register: The register that will be pointed to by the auxiliary register pointer (ARP) when an instruction that modifies ARP is finished executing. See also *auxiliary register*; *current auxiliary register*.

$\overline{\text{NMI}}$: A hardware interrupt that uses the same logic as the maskable interrupts but cannot be masked. It is often used as a soft reset. See also *maskable interrupt*; *nonmaskable interrupt*.

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.

NPAR: *Next program address register.* Part of the program-address generation logic. This register provides the address of the next instruction to the program counter (PC), the program address register (PAR), the micro stack (MSTACK), or the stack.

O

OE: *Receiver register overrun indicator bit*. Bit 9 of the I/O status register (IOSR); indicates whether overrun has occurred in the receiver of the asynchronous serial port (that is, whether an unread character in the ADTR has been overwritten by a new character).

operand: A value to be used or manipulated by an instruction; specified in the instruction.

operand-fetch phase: The third phase of the pipeline; the phase in which an operand or operands are fetched from memory. See also *pipeline*; *instruction-fetch phase*; *instruction-decode phase; instruction-execute phase*.

output shifter: 32- to 16-bit barrel left shifter. Shifts the 32-bit accumulator output from 0 to 7 bits left for quantization management, and outputs either the 16-bit high or low half of the shifted 32-bit data to the data write bus (DWEB).

OV bit: *Overflow flag bit.* Bit 12 of status register ST0; indicates whether the result of an arithmetic operation has exceeded the capacity of the accumulator.

overflow (in a register): A condition in which the result of an arithmetic operation exceeds the capacity of the register used to hold that result.

overflow (in the synchronous serial port): A condition in which the receive FIFO buffer of the port is full and another word is received in the RSR. (None of the contents of the FIFO buffer are overwritten by this new word.)

overflow mode: The mode in which an overflow in the accumulator will cause the accumulator to be loaded with a preset value. If the overflow is in the positive direction, the accumulator will be loaded with its most positive number. If the overflow is in the negative direction, the accumulator will be filled with its most negative number.

overrun: A condition in the receiver of the asynchronous serial port. Overrun occurs when an unread character in the ADTR is overwritten by a new character.

OVF bit: *Overflow bit (synchronous serial port).* Bit 7 of the synchronous serial port control register (SSPCR); indicates when the receive FIFO buffer of the port is full and another word is received in the RSR. (None of the contents of the FIFO buffer are overwritten by this new word.)

OVM bit: *Overflow mode bit.* Bit 11 of status register ST0; enables or disables overflow mode. See also *overflow mode*.

P

PAB: See *program address bus (PAB).*

PAR: *Program address register.* A register that holds the address currently being driven on the program address bus for as many cycles as it takes to complete all memory operations scheduled for the current machine cycle.

PC: See *program counter (PC).*

PCB: *Printed circuit board*.

pending interrupt: A maskable interrupt that has been successfully requested but is awaiting acknowledgement by the CPU.

period register: See *PRD*.

pipeline: A method of executing instructions in an assembly line fashion. The ’C2xx pipeline has four independent phases. During a given CPU cycle, four different instructions can be active, each at a different stage of completion. See also *instruction-fetch phase*; *instruction-decode phase*; *operand-fetch phase; instruction-execute phase*.

PLL: Phase lock loop circuit.

PM bits: See *product shift mode bits (PM).*

power-down mode: The mode in which the processor enters a dormant state and dissipates considerably less power than during normal operation. This mode is initiated by the execution of an IDLE instruction. During a power-down mode, all internal contents are maintained so that operation continues unaltered when the power-down mode is terminated. The contents of all on-chip RAM also remains unchanged.

PRD: *Timer period register*. A 16-bit memory-mapped register that specifies the main period for the on-chip timer. When the timer counter register (TIM) is decremented past zero, the TIM is loaded with the value in the PRD. See also *TDDR*.

PRDB: See *program read bus (PRDB).*

PREG: See *product register (PREG).*

prescaler counter: See *PSC.*

product register (PREG): A 32-bit register that holds the results of a multiply operation.

product shifter: A 32-bit shifter that performs a 0-, 1-, or 4-bit left shift, or a 6-bit right shift of the multiplier product based on the value of the product shift mode bits (PM).

product shift mode: One of four modes (no-shift, shift-left-by-one, shift-left-by-four, or shift-right-by-six) used by the product shifter.

product shift mode bits (PM): Bits 0 and 1 of status register ST1; they identify which of four shift modes (no-shift, left-shift-by-one, left-shift-by-four, or right-shift-by-six) will be used by the product shifter.

program address bus (PAB): A 16-bit internal bus that provides the addresses for program-memory reads and writes.

program-address generation logic: Logic circuitry that generates the addresses for program memory reads and writes, and an operand address in instructions that require two registers to address operands. This circuitry can generate one address per machine cycle. See also *data-address generation logic.*

program control logic: Logic circuitry that decodes instructions, manages the pipeline, stores status of operations, and decodes conditional operations.

program counter (PC): A register that indicates the location of the next instruction to be executed.

program read bus (PRDB): A 16-bit internal bus that carries instruction code and immediate operands, as well as table information, from program memory to the CPU.

$\overline{\text{PS}}$: *Program select pin*. The ’C2xx asserts $\overline{\text{PS}}$ to indicate an access to external program memory.

PSC: *Timer prescaler counter*. Bits 9–6 of the timer control register (TCR); specifies the prescale count for the on-chip timer.

PSLWS: *Lower program-space wait-state bits.* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip lower program space (addresses 0000h–7FFFh). PSLWS is not available on the ’C209; instead, see *PSWS*. On other ’C2xx devices, PSLWS is bits 2–0 of the WSGR. See also *PSUWS*.

PSUWS: *Upper program-space wait-state bits.* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip upper program space (addresses 8000h–FFFFh). PSUWS is not available on the ’C209; instead, see *PSWS*. On other ’C2xx devices, PSUWS is bits 5–3 of the WSGR. See also *PSLWS*.

PSWS: *Program-space wait-state bit.* Bit 0 of the ’C209 wait-state generator control register (WSGR). PSWS determines the number of wait states applied to reads from off-chip program memory space.

R

RAMEN: *RAM enable pin*. This pin enables or disables on-chip single-access RAM.

$\overline{RD}$: *Read select pin*. The ’C2xx asserts $\overline{RD}$ to request a read from external program, data, or I/O space. $\overline{RD}$ can be connected directly to the output enable pin of an external device.

READY: *External device ready pin*. Used to create wait states externally. When this pin is driven low, the ’C2xx waits one CPU cycle and then tests READY again. After READY is driven low, the ’C2xx does not continue processing until READY is driven high.

receive interrupt (asynchronous serial port): An interrupt (TXRXINT) caused during reception by any one of these events: the ADTR holds a new character; overrun occurs; a framing error occurs; a break has been detected on the RX pin; a character *A* or *a* has been detected in the ADTR by the automatic baud-rate detection logic.

receive interrupt (synchronous serial port): See *RINT.*

receive interrupt mask bit (RIM): Bit 7 of the asynchronous serial port control register (ASPCR); enables or disables receive interrupts of the asynchronous serial port.

receive pin (asynchronous serial port): See *RX pin*.

receive pin (synchronous serial port): See *DR pin*.

receive register (asynchronous serial port): See *ADTR*.

receive register (synchronous serial port): See *SDTR*.

receive reset (RRST) bit: Bit 4 of the synchronous serial port control register (SSPCR); resets the receiver portion of the synchronous serial port.

receive shift register (asynchronous serial port): See *ARSR*.

receive shift register (synchronous serial port): See *RSR.*

repeat counter (RPTC): A 16-bit register that counts the number of times a single instruction is repeated. RPTC is loaded by an RPT instruction.

reset: A way to bring the processor to a known state by setting the registers and control bits to predetermined values and signaling execution to start at address 0000h.

reset pin ($\overline{RS}$, also RS on ’C209): This pin causes a reset.

reset vector: The interrupt vector for reset.

return address: The address of the instruction to be executed when the CPU returns from a subroutine or interrupt service routine.

RFNE bit: *Receive FIFO buffer not empty bit*. Bit 12 of the synchronous serial port control register (SSPCR); indicates whether the receive FIFO buffer of the synchronous serial port contains data to be read.

RIM bit: See *receive interrupt mask bit (RIM).*

RINT: *Receive interrupt (synchronous serial port)*. An interrupt (RINT) generated during reception based on the number of words in the receive FIFO buffer. The trigger condition (the desired number of words in the buffer) is determined by the values of the receive-interrupt bits (FR1 and FR0) of the synchronous serial port control register (SSPCR).

RPTC: See *repeat counter (RPTC).*

RRST: *Receive reset bit*. Bit 4 of the synchronous serial port control register (SSPCR); resets the receiver portion of the synchronous serial port.

$\overline{\text{RS}}$: *Reset pin*. When driven low, causes a reset on any ’C2xx device, including the ’C209.

RS: *Reset pin*. (On the ’C209 only) When driven high, causes a reset.

RSR: *Receive shift register.* Shifts data serially into the synchronous serial port from the DR pin. See also *XSR*.

R/$\overline{\text{W}}$: *Read/write pin.* Indicates the direction of transfer between the ’C2xx and external program, data, or I/O space.

RX pin: *Asynchronous receive pin*. During reception in the asynchronous serial port, this pin accepts a character one bit at a time, transferring it to the ARSR.

S

SARAM: *Single-access RAM*. RAM that can accessed (read from or written to) once in a single CPU cycle.

scratch-pad RAM: Another name for DARAM block B2 in data space (32 words).

SDTR: *Synchronous data transmit and receive register.* An I/O-mapped read/write register that sends data to the transmit FIFO buffer and extracts data from the receive FIFO buffer.

SETBRK: Bit 4 of the asynchronous serial port control register (ASPCR); selects the output level (high or low) on the TX pin when the port is not transmitting.

short-immediate value: An 8-, 9-, or 13-bit constant given as an operand of an instruction that is using immediate addressing.

sign bit: The MSB of a value when it is seen by the CPU to indicate the sign (negative or positive) of the value.

sign extend: Fill the unused high order bits of a register with copies of the sign bit in that register.

sign-extension mode (SXM) bit: Bit 10 of status register ST1; enables or disables sign extension in the input shifter. It also differentiates between logic and arithmetic shifts of the accumulator.

single-access RAM: See *SARAM*.

slave phase: See *latch phase*.

SOFT bit (asynchronous serial port): Bit 14 in the asynchronous serial port control register (ASPCR); a special emulation bit that is used in conjunction with bit 15 (FREE) to determine the state of an asynchronous serial port transfer when a software breakpoint is encountered during emulation. When FREE = 0, SOFT determines the emulation mode. See also *FREE bit (asynchronous serial port)*.

SOFT bit (synchronous serial port): Bit 14 of the synchronous serial port control register (SSPCR); a special emulation bit that is used in conjunction with bit 15 (FREE) to determine the state of a synchronous serial port transfer when a software breakpoint is encountered during emulation. When FREE = 0, SOFT determines the emulation mode. See also *FREE bit (synchronous serial port)*.

SOFT bit (timer): Bit 10 of the timer control register (TCR); a special emulation bit that is used in conjunction with bit 11 (FREE) to determine the state of the timer when a software breakpoint is encountered during emulation. When FREE = 0, SOFT determines the emulation mode. SOFT and FREE are not available in the TCR of the ’C209. See also *FREE bit (timer)*.

software interrupt: An interrupt caused by the execution of an INTR, NMI, or TRAP instruction.

software stack: A program control feature that allows you to extend the hardware stack into data memory with the PSHD and POPD instructions. The stack can be directly stored and recovered from data memory, one word at time. This feature is useful for deep subroutine nesting or protection against stack overflow.

SSPCR: *Synchronous serial port control register.* A 16-bit I/O-mapped register that you write to when setting the configuration of the synchronous serial port and that you read when obtaining the status of the port.

ST0 and ST1: See *status registers ST0 and ST1*.

stack: A block of memory reserved for storing return addresses for subroutines and interrupt service routines. The 'C2xx stack is 16 bits wide and eight levels deep.

start bit: Every 8-bit data value transmitted or received by the asynchronous serial port must be preceded by a start bit, a logic 0 pulse.

status registers ST0 and ST1: Two 16-bit registers that contain bits for determining processor modes, addressing pointer values, and indicating various processor conditions and arithmetic logic results. These registers can be stored into and loaded from data memory, allowing the status of the machine to be saved and restored for subroutines.

STB bit: *Stop bit selector*. Bit 6 of the asynchronous serial port control register (ASPCR); selects the number of stop bits (one or two) used in transmission and reception.

stop bit: Every 8-bit data value transmitted or received by the asynchronous serial port must be followed by one or two stop bits, each a logic 1 pulse. The number of stop bits required depends on the STB bit of the ASPCR.

$\overline{\text{STRB}}$: *External access active strobe*. The 'C2xx asserts $\overline{\text{STRB}}$ during accesses to external program, data, or I/O space.

SXM bit: See *sign-extension mode bit (SXM).*

T

TC bit: *Test/control flag bit*. Bit 11 of status register ST1; stores the results of test operations done in the central arithmetic logic unit (CALU) or the auxiliary register arithmetic unit (ARAU). The TC bit can be tested by conditional instructions.

TCOMP: *Transmission complete bit*. Bit 13 of the synchronous serial port control register (SSPCR); indicates when all data in the transmit FIFO buffer of the synchronous serial port has been transmitted.

TCR: *Timer control register.* A 16-bit register that controls the operation of the on-chip timer.

TDDR: See *timer divide-down register (TDDR)*.

temporary register (TREG): A 16-bit register that holds one of the operands for a multiply operation; the dynamic shift count for the LACT, ADDT, and SUBT instructions; or the dynamic bit position for the BITT instruction.

TEMT bit: *Transmit empty indicator*. Bit 12 of the I/O status register (IOSR); indicates whether the transmit register (ADTR) and/or the transmit shift register (AXSR) of the asynchronous serial port are full or empty.

THRE bit: *Transmit register empty indicator*. Bit 11 of the I/O status register (IOSR); indicates when the contents of the transmit register (ADTR) are transferred to the transmit shift register (AXSR).

TIM bit: *Transmit interrupt mask bit*. Bit 8 of the asynchronous serial port control register (ASPCR); enables or disables transmit interrupts of the asynchronous serial port.

TIM register: See *timer counter register (TIM)*.

timer counter register (TIM): A 16-bit memory-mapped register that holds the main count for the on-chip timer. See also *timer prescaler counter (PSC)*.

timer divide-down register (TDDR): Bits 3–0 of the timer control register (TCR); specifies the timer divide-down period for the on-chip timer. When the timer prescaler counter (PSC) decrements past zero, the PSC is loaded with the value in the TDDR. See also *timer period register (PRD)*.

timer interrupt (TINT): See *TINT*.

timer period register (PRD): A 16-bit memory-mapped register that specifies the main period for the on-chip timer. When the timer counter register (TIM) is decremented past zero, the TIM is loaded with the value in the PRD. See also *TDDR*.

timer prescaler counter (PSC): Bits 9–6 of the timer control register (TCR); specifies the prescale count for the on-chip timer.

timer reload bit (TRB): Bit 5 of the timer control register (TCR); when TRB is set, the timer counter register (TIM) is loaded with the value of the timer period register (PRD), and the prescaler counter (PSC) is loaded with the value of the timer divide-down register (TDDR).

timer stop status bit (TSS): Bit 4 of the TCR. TSS is used to start and stop the timer.

TINT: *Timer interrupt*. An interrupt generated by the timer on the next CLKOUT1 cycle after the main counter (TIM register) decrements to 0

TOS: *Top of stack.* Top level of the 8-level last-in, first-out hardware stack.

TOUT: *Timer output pin.* Provides access to an output signal based on the rate of the on-chip timer. On the next CLKOUT1 cycle after the main counter (TIM register) decrements to 0, a signal is sent to TOUT.

transmit interrupt (asynchronous serial port): An interrupt (TXRXINT) generated when the transmit register (ADTR) empties during transmission. This condition indicates that the ADTR is ready to accept a new transmit character.

transmit interrupt (synchronous serial port): See *XINT*.

transmit mode (TXM) bit: Bit 3 of the synchronous serial port control register (SSPCR); determines whether the source signal for frame synchronization is external or internal.

transmit pin (asynchronous serial port): See *TX pin*.

transmit pin (synchronous serial port): See *DX pin*.

transmit/receive interrupt (TXRXINT): The CPU interrupt used to respond to a delta interrupt, receive interrupt, or transmit interrupt from the asynchronous serial port. All three of these interrupt types request TXRXINT and use the single TXRXINT interrupt vector. See also *delta interrupt*; *receive interrupt*; *transmit interrupt*.

transmit register (asynchronous serial port): See *ADTR*.

transmit register (synchronous serial port): See *SDTR*.

transmit reset (XRST) bit: Bit 5 of the synchronous serial port control register (SSPCR); resets the transmitter portion of the synchronous serial port.

transmit shift register (asynchronous serial port): Also called AXSR, this register shifts data serially out of the asynchronous serial port through the TX pin. See also *ARSR*.

transmit shift register (synchronous serial port): Also called XSR, this register shifts data serially out of the synchronous serial port through the DX pin. See also *RSR*.

TRB: See *timer reload bit (TRB).*

TREG: See *temporary register (TREG).*

TSS bit: See *timer stop status bit (TSS).*

TTL: *Transistor-to-transistor logic*.

TX pin: *Asynchronous transmit pin*. The pin on which data is transmitted serially from the asynchronous serial port; accepts a character one bit at a time from the transmit shift register (AXSR).

TXM: *Transmit mode bit*. Bit 3 of the synchronous serial port control register (SSPCR); determines whether the source signal for frame synchronization is external or internal.

TXRXINT: See *transmit/receive interrupt (TXRXINT)*.

U

UART: *Universal asynchronous receiver and transmitter.* Used as another name for the asynchronous serial port.

URST: *Reset asynchronous serial port bit*. Bit 13 of the asynchronous serial port control register (ASPCR); resets the asynchronous port.

V

vector: See *interrupt vector*.

vector location: See *interrupt vector location*.

W

wait state: A CLKOUT1 cycle during which the CPU waits when reading from or writing to slower external memory.

wait-state generator: An on-chip peripheral that generates a limited number of wait states for a given off-chip memory space (program, data, or I/O). Wait states are set in the wait-state generator control register (WSGR).

$\overline{\text{WE}}$: *Write enable pin*. The ’C2xx asserts $\overline{\text{WE}}$ to request a write to external program, data, or I/O space.

WSGR: *Wait-state generator control register.* This register, which is mapped to I/O memory, controls the wait-state generator.

X

XF bit: *XF-pin status bit.* Bit 4 of status register ST1 that is used to read or change the logic level on the XF pin.

XF pin: *External flag pin*. A general-purpose output pin whose status can be read or changed by way of the XF bit in status register ST1.

XINT: *Transmit interrupt (synchronous serial port)*. An interrupt generated during transmission based on the number of words in the transmit FIFO buffer. The trigger condition (the desired number of words in the buffer) is determined by the values of the transmit-interrupt bits (FT1 and FT0) of the synchronous serial port control register (SSPCR).

XRST: *Transmit reset bit*. Bit 5 of the synchronous serial port control register (SSPCR); resets the transmitter portion of the synchronous serial port.

XSR: *Transmit shift register.* Shifts data serially out of the synchronous serial port through the DX pin. See also *RSR*.

Z

zero fill: Fill the unused low or high order bits in a register with zeros.

Index

* operand 6-10
*+ operand 6-10
*– operand 6-10
*0+ operand 6-10
*0– operand 6-10
*BR0+ operand 6-11
*BR0– operand 6-11
14-pin connector
    dimensions F-15
14-pin header
    header signals F-2
    JTAG F-2
4-level pipeline operation 5-7

A

A0–A15 (external address bus)
    definition 4-3
    shown in figure 4-6, 4-10, 4-13, 4-17, 4-31
ABS instruction 7-21
absolute value (ABS instruction) 7-21
accumulator
    definition G-1
    description 3-9
    shifting and storing high and low words, diagrams 3-11
accumulator instructions
    absolute value of accumulator (ABS) 7-21
    add PREG to accumulator (APAC) 7-37
    add PREG to accumulator and load TREG (LTA) 7-93
    add PREG to accumulator and multiply (MPYA) 7-116
    add PREG to accumulator and square specified value (SQRA) 7-168
    add PREG to accumulator, load TREG, and move data (LTD) 7-95
accumulator instructions (continued)
    add PREG to accumulator, load TREG, and multiply (MAC) 7-102
    add PREG to accumulator, load TREG, multiply, and move data (MACD) 7-106
    add value plus carry to accumulator (ADDC) 7-27
    add value to accumulator (ADD) 7-23
    add value to accumulator with shift specified by TREG (ADDT) 7-31
    add value to accumulator with sign extension suppressed (ADDS) 7-29
    AND accumulator with value (AND) 7-34
    branch to location specified by accumulator (BACC) 7-40
    call subroutine at location specified by accumulator (CALA) 7-58
    complement accumulator (CMPL) 7-64
    divide using accumulator (SUBC) 7-180
    load accumulator (LACC) 7-74
    load accumulator using shift specified by TREG (LACT) 7-78
    load accumulator with PREG (PAC) 7-134
    load accumulator with PREG and load TREG (LTP) 7-98
    load high bits of accumulator with rounding (ZALR) 7-196
    load low bits and clear high bits of accumulator (LACL) 7-75
    negate accumulator (NEG) 7-122
    normalize accumulator (NORM) 7-126
    OR accumulator with value (OR) 7-129
    pop top of stack to low accumulator bits (POP) 7-135
    push low accumulator bits onto stack (PUSH) 7-141
    rotate accumulator left by one bit (ROL) 7-144
    rotate accumulator right by one bit (ROR) 7-145
    shift accumulator left by one bit (SFL) 7-157
    shift accumulator right by one bit (SFR) 7-158

accumulator instructions (continued)
- store high byte of accumulator to data memory (SACH) 7-148
- store low byte of accumulator to data memory (SACL) 7-150
- subtract conditionally from accumulator (SUBC) 7-180
- subtract PREG from accumulator (SPAC) 7-160
- subtract PREG from accumulator and load TREG (LTS) 7-100
- subtract PREG from accumulator and multiply (MPYS) 7-118
- subtract PREG from accumulator and square specified value (SQRS) 7-170
- subtract value and logical inversion of carry bit from accumulator (SUBB) 7-178
- subtract value from accumulator (SUB) 7-174
- subtract value from accumulator with shift specified by TREG (SUBT) 7-184
- subtract value from accumulator with sign extension suppressed (SUBS) 7-182
- XOR accumulator with data value (XOR) 7-193

ADC bit 10-10

ADD instruction 7-23

ADDC instruction 7-27

address generation
- data memory
  - *direct addressing 6-4*
  - *immediate addressing 6-2*
  - *indirect addressing 6-9*
- program memory 5-2
  - *hardware 5-3*

address maps
- ’C203 4-23
- ’C209 11-6
- data page 0 4-8

address visibility mode (AVIS bit) 11-18

addressing
- bit-reversed indexed 6-10, G-3

addressing modes
- definition G-1
- direct
  - *description 6-4*
  - *examples 6-6*
  - *figure 6-5*
  - *opcode format 6-5 to 6-7*
  - *role of data page pointer (DP) 6-4*
- immediate 6-2

addressing modes (continued)
- indirect
  - *description 6-9*
  - *effects on auxiliary register pointer (ARP) 6-14 to 6-16*
  - *effects on current auxiliary register 6-14 to 6-16*
  - *examples 6-15*
  - *modifying auxiliary register content 6-17*
  - *opcode format 6-12 to 6-14*
  - *operands 6-9*
  - *operation types 6-14 to 6-16*
  - *options 6-9*
  - *possible opcodes 6-14 to 6-16*
- overview 6-1

ADDS instruction 7-29

ADDT instruction 7-31

ADRK instruction 7-33

ADTR (asynchronous serial port transmit and receive register) 10-4

AND instruction 7-34

APAC instruction 7-37

applications
- TMS320 devices 1-3

ARAU (auxiliary register arithmetic unit) 3-12

ARAU and related logic
- block diagram 3-12

ARB (auxiliary register pointer buffer) 3-16

architecture of ’C2xx 2-1 to 2-14

arithmetic logic unit
- central (CALU) 3-9

ARP (auxiliary register pointer) 3-16

ARSR (asynchronous serial port receive shift register) 10-5

ASPCR (asynchronous serial port control register) 10-7
- configuring pins IO0–IO3 as inputs/outputs 10-16
- quick reference A-17

asynchronous
- reception 10-20
- transmission 10-19

asynchronous serial port
- basic operation 10-5
- baud rates
  - *common 10-14*
  - *setting 10-14*

asynchronous serial port (continued)
- baud-rate detection logic
  - *detecting A or a character (ADC bit) 10-10*
  - *enabling/disabling (CAD bit) 10-8*
- block diagram 10-3
- components 10-3
- configuration 10-7
- delta interrupts 10-18
  - *enabling/disabling (DIM bit) 10-8*
- emulation modes (FREE and SOFT bits) 10-7
- features 10-1
- interrupts (TXRXINTs)
  - *flag bit (TXRXINT) 5-21*
  - *introduction 10-5*
  - *mask bit in IMR (TXRXINT) 5-23*
  - *mask bits in ASPCR (DIM, TIM, RIM) 10-8*
  - *priority 5-16*
  - *three types 10-17*
  - *vector location 5-16*
- introduction 2-12
- overrun in receiver, detecting (OE bit) 10-11
- overview 10-2
- receive interrupts 10-17
  - *enabling/disabling (RIM bit) 10-8*
- receive pin (RX)
  - *definition 10-4*
  - *detecting break on (BI bit) 10-10*
- receiver operation 10-20
- reset conditions 5-36
- resetting (URST bit) 10-8
- signals 10-3
  - *data 10-3*
  - *handshake 10-3*
- stop bit(s)
  - *detecting invalid (FE bit) 10-11*
  - *setting number of (STB bit) 10-8*
- transmit interrupts 10-17
  - *enabling/disabling (TIM bit) 10-8*
- transmit pin (TX)
  - *definition 10-4*
  - *output level between transmissions (SETBRK bit) 10-9*
- transmitter operation 10-19

asynchronous serial port registers
- baud-rate divisor register (BRD) 10-14
- control register (ASPCR) 10-7
  - *configuring pins IO0–IO3 as inputs/outputs 10-16*
  - *quick reference A-17*

asynchronous serial port registers (continued)
- I/O status register (IOSR)
  - *description 10-10*
  - *quick reference A-17*
- introduction 10-4
- receive register (ADTR)
  - *detecting overrun in (OE bit) 10-11*
  - *detecting when empty (DR bit) 10-12*
- receive shift register (ARSR) 10-5
- receive/transmit register (ADTR) 10-4
- transmit register (ADTR)
  - *detecting when empty (THRE bit) 10-11*
  - *detecting when it and AXSR are empty (TEMT bit) 10-11*
- transmit shift register (AXSR) 10-5
  - *detecting when it and ADTR are empty (TEMT bit) 10-11*
- transmit/receive register (ADTR) 10-4

automatic baud-rate detection 10-14

auxiliary register arithmetic unit (ARAU)
- description 3-12

auxiliary register instructions
- add short immediate value to current auxiliary register (ADRK) 7-33
- branch if current auxiliary register not zero (BANZ) 7-41
- compare current auxiliary register with AR0 (CMPR) 7-65
- load specified auxiliary register (LAR) 7-80
- modify auxiliary register pointer (MAR) 7-111
- modify current auxiliary register (MAR) 7-111
- store specified auxiliary register (SAR) 7-152
- subtract short immediate value from current auxiliary register (SBRK) 7-154

auxiliary register pointer (ARP) 3-16, G-2

auxiliary register pointer buffer (ARB) 3-16, G-2

auxiliary register update (ARU) code 6-13

auxiliary registers (AR0–AR7)
- block diagram 3-12
- current auxiliary register 6-9
  - *role in indirect addressing 6-9 to 6-18*
  - *update code (ARU) 6-13*
- description 3-12 to 3-14
- general uses for 3-14
- instructions that modify content 6-17
- next auxiliary register 6-11
- used in indirect addressing 3-12

AVIS bit 11-18

AXSR (asynchronous serial port transmit shift register) 10-5

B

B instruction 7-39

BACC instruction 7-40

BANZ instruction 7-41

baud-rate
- detection procedure 10-14
- divisor register (BRD) 10-14
- generator 10-4

BCND instruction 7-43

BI bit 10-10

$\overline{\text{BIO}}$ pin 8-18 to 8-19

BIT instruction 7-45

bit-reversed indexed addressing 6-10, G-3

BITT instruction 7-47

BLDD instruction 7-49

block diagrams
- ’C2xx overall 2-2
- ARAU and related logic 3-12
- arithmetic logic section of CPU 3-8
- asynchronous serial port 10-3
- auxiliary registers (AR0–AR7) and ARAU 3-12
- bus structure 2-4
- CPU (selected sections) 3-2
- input scaling section of CPU 3-3
- multiplication section of CPU 3-5
- program-address generation 5-2
- synchronous serial port 9-3
- timer 8-8

block move instructions
- block move from data memory to data memory (BLDD) 7-49
- block move from program memory to data memory (BLPD) 7-54

BLPD instruction 7-54

Boolean logic instructions
- AND 7-34
- CMPL (complement/NOT) 7-64
- OR 7-129
- XOR (exclusive OR) 7-193

$\overline{\text{BOOT}}$ (boot load pin)
- definition 4-4

bootloader 4-30 to 4-38
- boot source (EPROM)
  - *choosing an EPROM 4-30*
  - *connecting the EPROM 4-31*
  - *programming the EPROM 4-32*
- diagram 4-30 to 4-38
- enabling 4-33
- execution 4-34
- generating code for EPROM D-23 to D-24
- program code 4-37

$\overline{\text{BR}}$ (bus request pin)
- definition 4-3
- shown in figure 4-13, 4-31

branch instructions
- branch conditionally (BCND) 7-43
- branch if current auxiliary register not zero (BANZ) 7-41
- branch to location specified by accumulator (BACC) 7-40
- branch to $\overline{\text{NMI}}$ interrupt vector location (NMI) 7-124
- branch to specified interrupt vector location (INTR) 7-71
- branch to TRAP interrupt vector location (TRAP) 7-192
- branch unconditionally (B) 7-39
- call subroutine at location specified by accumulator (CALA) 7-58
- call subroutine conditionally (CC) 7-60
- call subroutine unconditionally (CALL) 7-59
- conditional, overview 5-11
- return conditionally from subroutine (RETC) 7-143
- return unconditionally from subroutine (RET) 7-142
- unconditional, overview 5-8

BRD (baud-rate divisor register) 10-14

buffered signals
- JTAG F-10

buffering F-10

burst mode
- definition G-3
- error conditions 9-27
- reception 9-22
- transmission
  - *with external frame sync 9-17*
  - *with internal frame sync 9-16*

bus devices F-4

bus protocol in emulator system F-4

bus request pin ($\overline{BR}$)
  definition 4-3
  shown in figure 4-13, 4-31
buses
  block diagram 2-4
  data read bus (DRDB) 2-3
  data write bus (DWEB) 2-3
  data-read address bus (DRAB) 2-3
  data-write address bus (DWAB) 2-3
  program address bus (PAB)
    definition 2-3
    used in program-memory address generation 5-3
  program read bus (PRDB) 2-3

C

C (carry bit)
  affected during SFL and SFR instructions 7-157 to 7-159
  definition 3-16
  involved in accumulator events 3-10
  used during ROL and ROR instructions 7-144 to 7-146
’C209 device 11-1 to 11-18
  comparison to other ’C2xx devices 11-2
    differences in interrupts 11-3
    differences in memory and I/O spaces 11-3
    differences in peripherals 11-2
    similarities 11-2
  interrupts 11-10
  locating ’C209 information in this manual (table) 11-3
  memory and I/O spaces 11-5
  on-chip peripherals 11-15
cable
  target system to emulator F-1 to F-25
cable pod F-5, F-6
CAD bit 10-8
CALA instruction 7-58
CALL instruction 7-59
call instructions
  call subroutine at location specified by accumulator (CALA) 7-58
  call subroutine conditionally (CC) 7-60
  call subroutine unconditionally (CALL) 7-59
  conditional, overview 5-12
  unconditional, overview 5-8
CALU (central arithmetic logic unit)
  definition G-4
  description 3-9
carry bit (C)
  affected during SFL and SFR instructions 7-157 to 7-159
  definition 3-16
  involved in accumulator events 3-10
  used during ROL and ROR instructions 7-144 to 7-146
CC instruction 7-60
central arithmetic logic section of CPU 3-8
CIO0–CIO3 (bits)
  configuring pins IO0–IO3 as inputs/outputs 10-16
CLK register
  description 8-7
  quick reference A-11
  reset condition 5-36
CLKIN signal 8-4 to 8-6
CLKMOD pin 11-15, G-4
CLKOUT1 bit 8-7
CLKOUT1 signal 8-4 to 8-6
  definition G-4
  turning CLKOUT1 pin on and off 8-7
CLKOUT1-pin control (CLK) register
  description 8-7
  quick reference A-11
  reset condition 5-36
CLKR pin
  as bit input (IN0 bit) 9-10
  definition 9-4
CLKX pin 9-4
clock generator 8-4 to 8-6
  ’C209 clock options 11-15 to 11-18
  introduction 2-11
  modes
    ’C203/C204 8-5
    ’C209 11-15 to 11-18
clock mode bit (MCM) 9-11
clock modes
  clock generator
    ’C203/C204 8-5
    ’C209 11-15
  synchronous serial port 9-11
CLRC instruction 7-62
CMPL instruction 7-64
CMPR instruction 7-65

CNF (DARAM configuration bit) 3-16

code compatibility 1-5

codec
  definition G-5

conditional instructions 5-10 to 5-13
  conditional branch 5-11 to 5-13
  conditional call 5-12 to 5-13
  conditional return 5-12 to 5-13
  conditions that may be tested 5-10
  stabilization of conditions 5-11
  using multiple conditions 5-10

configuration
  memory
    global data 4-11
    RAM (dual-access)
      ’C203 4-24
      ’C209 11-8
    RAM (single-access) 11-7
    ROM, ’C209 11-7
  multiprocessor F-13

connector
  14-pin header F-2
  dimensions, mechanical F-14
  DuPont F-2

continuous mode
  error conditions 9-27
  reception 9-23
  transmission
    with external frame sync 9-20
    with internal frame sync 9-19

control instructions (summary) 7-9

CPU 3-1 to 3-18
  accumulator 3-9
  arithmetic logic section 3-8
  auxiliary register arithmetic unit (ARAU) 3-12
  block diagram (partial) 3-2
  CALU (central arithmetic logic unit) 3-9
  central arithmetic logic unit (CALU) 3-9
  definition G-5
  input scaling section/input shifter 3-3
  key features 1-5
  multiplication section 3-5
  output shifter 3-11
  overview 2-5
  product shifter 3-6
    product shift modes 3-7
  status registers ST0 and ST1 3-15

current auxiliary register 6-9
  add short immediate value to (ADRK instruction) 7-33
  branch if not zero (BANZ instruction) 7-41
  compare with AR0 (CMPR instruction) 7-65
  increment or decrement (MAR instruction) 7-111
  role in indirect addressing 6-9 to 6-18
  subtract short immediate value from (SBRK instruction) 7-154
  update code (ARU) 6-13

D

D0–D15 (external data bus)
  definition 4-3, G-5
  shown in figure 4-6, 4-10, 4-13, 4-17, 4-31

DARAM (dual-access RAM)
  configuration
    ’C203 4-24
    ’C209 11-8
  description 2-7

DARAM configuration bit (CNF) 3-16

data memory
  address map
    ’C203 4-23
    ’C209 11-6
    data page 0 4-8
  caution about reserved addresses 4-24, 11-7
  configuration
    RAM (dual-access)
      ’C203 4-24
      ’C209 11-8
    RAM (single-access) 11-7
  data page pointer (DP) 3-16
  external interfacing
    caution about proper timing 4-9
    global 4-13
    local 4-9
  global 4-11
  local 4-7
  on-chip registers mapped to 4-8

data memory select pin ($\overline{DS}$)
  definition 4-3
  shown in figure 4-10, 4-13

data page 0 4-8
  caution about test/emulation addresses 4-8

data page pointer (DP)
- caution about initializing DP 6-5
- definition 3-16
- load (LDP instruction) 7-83
- role in direct addressing 6-4

data read bus (DRDB) 2-3

data write bus (DWEB) 2-3

data-read address bus (DRAB) 2-3

data-scaling shifter
- at input of CALU 3-3
- at output of CALU 3-11

data-write address bus (DWAB) 2-3

delta interrupts
- description 10-18
- enabling/disabling (DIM bit) 10-8

device reset 5-35

diagnostic applications F-24

digital loopback mode 9-26

DIM bit 10-8

dimensions
- 12-pin header F-20
- 14-pin header F-14
- mechanical, 14-pin header F-14

DIO0–DIO3 (bits)
- detecting change on pins IO0–IO3 10-17

direct addressing
- description 6-4
- examples 6-6
- figure 6-5
- opcode format 6-5 to 6-7
- role of data page pointer (DP) 6-4

direct memory access (using HOLD operation) 4-18
- during reset 4-20
- example 4-19
- terminating correctly 4-20

DIV1 and DIV2 pins 8-5, G-7

divide (SUBC instruction) 7-180

DLB bit 9-11

DMOV instruction 7-66

DP (data page pointer)
- caution about initializing DP 6-5
- definition 3-16
- load (LDP instruction) 7-83
- role in direct addressing 6-4

DR bit 10-12

DR pin 9-4

DRAB (data-read address bus) 2-3

DRDB (data read bus) 2-3

$\overline{\text{DS}}$ (data memory select pin)
- definition 4-3
- shown in figure 4-10, 4-13

DSWS bit(s)
- ’C203/C204 8-16
- ’C209 11-18

dual-access RAM (DARAM) G-5
- configuration
  - *’C203 4-24*
  - *’C209 11-8*
- description 2-7

DuPont connector F-2

DWAB (data-write address bus) 2-3

DWEB (data write bus) 2-3

DX pin 9-4

E

EMU0/1
- configuration F-21, F-23, F-24
- emulation pins F-20
- IN signals F-21
- rising edge modification F-22

EMU0/1 signals F-2, F-3, F-6, F-7, F-13, F-18

emulation
- configuring multiple processors F-13
- JTAG cable F-1
- pins F-20
- timing calculations F-7 to F-9, F-18 to F-26
- using scan path linkers F-16

emulation capability 2-13

emulation modes (FREE and SOFT bits)
- asynchronous serial port 10-7
- synchronous serial port 9-9

emulation timing F-7

emulator
- cable pod F-5
- connection to target system, JTAG mechanical dimensions F-14 to F-25
- designing the JTAG cable F-1
- emulation pins F-20
- pod interface F-5
- pod timings F-6
- signal buffering F-10 to F-13
- target cable, header design F-2 to F-3

enhanced instructions B-3, C-5

error conditions
  asynchronous serial port
    framing error (FE bit)   10-11
    overrun (OE bit)   10-11
  synchronous serial port
    burst mode   9-27
    continuous mode   9-27
examples of ’C2xx program code   D-1 to D-24
external access active strobe (STRB)   4-3
external address bus (A0–A15)
  definition   4-3
  shown in figure   4-6, 4-10, 4-13, 4-17, 4-31
external data bus (D0–D15)
  definition   4-3
  shown in figure   4-6, 4-10, 4-13, 4-17, 4-31
external device ready pin (READY)
  definition   4-4
  generating wait states with   8-15
external interfacing
  diagrams   4-6, 4-10, 4-13, 4-17
external oscillator
  using (diagram)   8-5

F

FE bit   10-11
features summary   1-5
FIFO buffers
  introduction   9-5
FINT2 bit   5-26
FINT3 bit   5-26
flag bits
  I/O status register (IOSR)   10-10
  interrupt control register (ICR)   5-18
  interrupt flag register (IFR)   5-18
flash memory (on-chip)
  introduction   2-9
flow charts
  interrupt operation
    maskable interrupts   5-20
    nonmaskable interrupts   5-29
    requesting INT2 and INT3   5-18
  TMS320 ROM code submittal   E-2
FR1 and FR0 bits   9-10
frame synchronization mode (FSM bit)   9-11
framing error (FE bit)   10-11
FREE bit   9-9
  asynchronous serial port   10-7
  timer   8-11
FSM bit   9-11
FSR pin   9-4
FSX pin   9-4
FT1 and FT0 bits   9-9

G

general-purpose I/O pins   8-18 to 8-21
  input
    BIO   8-18 to 8-19
    IO0–IO3   10-15 to 10-16
  output
    IO0–IO3   10-15 to 10-16, 10-17
    XF   8-19
generating executable files
  figure   D-2
generating wait states with   8-15
generators (on-chip)
  baud-rate generator   10-4
  clock generator   8-4 to 8-6
    ’C209 clock options   11-15 to 11-18
  wait-state generator   8-15 to 8-17
    ’C209   11-17 to 11-18
global data memory   4-11
  configuration   4-11
  external interfacing   4-13
global memory allocation register (GREG)   4-11
GREG (global memory allocation register)   4-11

H

hardware interrupts
  definition   5-15
  nonmaskable external   5-27
  priorities   5-16
  types   5-15
hardware reset   5-35
header
  14-pin   F-2
  dimensions, 14-pin   F-2
HOLD (HOLD operation request pin)
  definition   4-4
  use in HOLD operation   4-18
HOLD acknowledge pin (HOLDA)
  definition   4-4
  use in HOLD operation   4-18

HOLD operation
- description 4-18
- during reset 4-20
- example 4-19
- terminating correctly 4-20

HOLD operation request pin ($\overline{\text{HOLD}}$)
- definition 4-4
- use in HOLD operation 4-18

HOLD/INT1 bit
- in interrupt flag register (IFR) 5-22
- in interrupt mask register (IMR) 5-24

$\overline{\text{HOLD}}/\overline{\text{INT1}}$ interrupt
- flag bit 5-22
- mask bit 5-24
- priority 5-16
- vector location 5-16

$\overline{\text{HOLD}}/\overline{\text{INT1}}$ pin
- mode set by MODE bit 5-24

$\overline{\text{HOLDA}}$ ($\overline{\text{HOLD}}$ acknowledge pin)
- definition 4-4
- use in HOLD operation 4-18

I

I/O
- general-purpose pins
  - *input*
    - $\overline{\text{BIO}}$ 8-18 to 8-19
    - IO0–IO3 10-15 to 10-17
  - *output*
    - IO0–IO3 10-15 to 10-17
    - XF 8-19
- parallel ports 4-16
- serial ports
  - *asynchronous 10-1 to 10-20*
  - *introduction 2-12*
  - *synchronous 9-1 to 9-42*

I/O space
- accessing 4-16
- address map 4-14
- caution about reserved addresses 4-15
- description 4-14
- external interfacing 4-16
- instructions
  - *transfer data from data memory to I/O space (OUT) 7-132*
  - *transfer data from I/O space to data memory (IN) 7-69*

I/O space (continued)
- on-chip registers mapped to
  - *'C203/C204 4-16*
  - *'C209 11-9*
  - *accessing 4-16*
- pins for external interfacing 4-3

I/O space select pin ($\overline{\text{IS}}$)
- definition 4-3
- shown in figure 4-17

I/O status register (IOSR)
- description 10-10
- detecting change on pins IO0–IO3 10-17
- quick reference A-18
- reading current logic level on pins IO0–IO3 10-17

I/O-mapped registers
- addresses and reset values A-2

$\overline{\text{IACK}}$ signal 11-13

ICR (interrupt control register) 5-24 to 5-42
- bits 5-26
- quick reference A-8

IDLE instruction 7-68

IEEE 1149.1 specification
- bus slave device rules F-4

IFR (interrupt flag register) 5-20 to 5-42
- bits
  - *'C203/C204 5-21*
  - *'C209 11-12*
- clearing interrupts 5-20
- quick reference A-6

immediate addressing 6-2

IMR (interrupt mask register) 5-23 to 5-42
- bits
  - *'C203/C204 5-23*
  - *'C209 11-13*
- in interrupt acknowledgement process 5-19
- quick reference A-7

IN instruction 7-69

IN0 bit 9-10

indirect addressing
- description 6-9
- effects on auxiliary register pointer (ARP) 6-14 to 6-16
- effects on current auxiliary register 6-14 to 6-16
- examples 6-15
- modifying auxiliary register content 6-17
- opcode format 6-12 to 6-14
- operands 6-10

indirect addressing (continued)
- operation types 6-14 to 6-16
- options 6-9
- possible opcodes 6-14 to 6-16

input clock modes
- 'C203/C204 8-5
- 'C209 11-15

input scaling section of CPU 3-3

input shifter 3-3

input/output status register (IOSR)
- description 10-10
- detecting change on pins IO0–IO3 10-17
- reading current logic level on pins IO0–IO3 10-17

instruction register (IR)
- definition G-11

instruction set
- key features 1-6

instructions 7-1 to 7-20
- Boolean logic
  - *AND 7-34*
  - *CMPL (complement/NOT) 7-64*
  - *OR 7-129*
  - *XOR (exclusive OR) 7-193*
- compared with those of other TMS320 devices B-1 to B-10, C-1 to C-36
- conditional 5-10 to 5-13
  - *branch (BCND) 7-43*
  - *call (CC) 7-60*
  - *conditions that may be tested 5-10*
  - *return (RETC) 7-143*
  - *stabilization of conditions 5-11*
  - *using multiple conditions 5-10*
- CPU halt until hardware interrupt (IDLE) 7-68
- delay/no operation (NOP) 7-125
- descriptions 7-20
  - *how to use 7-12*
- enhanced B-3, C-5
- idle until hardware interrupt (IDLE) 7-68
- interrupt
  - *branch to $\overline{NMI}$ interrupt vector location (NMI) 7-124*
  - *branch to specified interrupt vector location (INTR) 7-71*
  - *branch to TRAP interrupt vector location (TRAP) 7-192*

instructions (continued)
- negate accumulator (NEG) 7-122
- no operation (NOP) 7-125
- normalize (NORM) 7-126
- OR 7-129
- power down until hardware interrupt (IDLE) 7-68
- repeat next instruction n times
  - *description (RPT) 7-146*
  - *introduction 5-14*
- stack
  - *pop top of stack to data memory (POPD) 7-137*
  - *pop top of stack to low accumulator bits (POP) 7-135*
  - *push data memory value onto stack (PSHD) 7-139*
  - *push low accumulator bits onto stack (PUSH) 7-141*
- status registers ST0 and ST1
  - *clear control bit (CLRC) 7-62*
  - *load (LST) 7-87*
  - *load data page pointer (LDP) 7-83*
  - *modify auxiliary register pointer (MAR) 7-111*
  - *set control bit (SETC) 7-155*
  - *set product shift mode (SPM) 7-167*
  - *store (SST) 7-172*
- summary 7-2 to 7-11
- test bit specified by TREG (BITT) 7-47
- test specified bit (BIT) 7-45

INT1 bit ('C209)
- in interrupt flag register (IFR) 11-12
- in interrupt mask register (IMR) 11-13

$\overline{INT1}$ interrupt
- 'C203/C204
  - *flag bit (HOLD/INT1) 5-22*
  - *mask bit (HOLD/INT1) 5-24*
  - *priority 5-16*
  - *vector location 5-16*
- 'C209
  - *flag bit 11-12*
  - *mask bit 11-13*
  - *priority 11-10*
  - *vector location 11-10*

INT2 bit ('C209)
- in interrupt flag register (IFR) 11-12
- in interrupt mask register (IMR) 11-13

$\overline{\text{INT2}}$ interrupt
- 'C203/C204
  - *flag bits*
    - FINT2 5-26
    - INT2/INT3 5-22
  - *masking/unmasking in ICR 5-27*
  - *masking/unmasking in IMR 5-24*
  - *priority 5-16*
  - *vector location 5-16*
- 'C209
  - *flag bit 11-12*
  - *mask bit 11-13*
  - *priority 11-10*
  - *vector location 11-10*

INT2/INT3 bit
- in interrupt flag register (IFR) 5-22
- in interrupt mask register (IMR) 5-24

INT20–INT31 (interrupts), vector locations
- 'C203/C204 5-17
- 'C209 11-11

INT3 bit ('C209)
- in interrupt flag register (IFR) 11-12
- in interrupt mask register (IMR) 11-13

$\overline{\text{INT3}}$ interrupt
- 'C203/C204
  - *flag bits*
    - FINT3 5-26
    - INT2/INT3 5-22
  - *masking/unmasking in ICR 5-27*
  - *masking/unmasking in IMR 5-24*
  - *priority 5-16*
  - *vector location 5-16*
- 'C209
  - *flag bit 11-12*
  - *mask bit 11-13*
  - *priority 11-10*
  - *vector location 11-10*

INT8–INT16 (interrupts), vector locations
- 'C203/C204 5-16 to 5-17
- 'C209 11-10

interfacing
- to external global data memory 4-13
- to external I/O space 4-16
- to external local data memory 4-9
- to external program memory 4-5

internal oscillator
- using (diagram) 8-4

interrupt 5-15 to 5-34
- definitions 5-15, G-11
- hardware interrupt
  - *definition 5-15*
  - *priorities*
    - 'C203/C204 5-16
    - 'C209 11-10
- interrupt mode bit (INTM) 3-16
  - *use in enabling/disabling maskable interrupts 5-19*
- interrupt service routines (ISRs) 5-29 to 5-30
  - *ISRs within ISRs 5-30*
  - *saving and restoring context 5-29 to 5-30*
- latency 5-31 to 5-32
  - *after execution of RET 5-32*
  - *during execution of CLRC INTM 5-32*
  - *minimum latency 5-31*
- maskable interrupt 5-18 to 5-20
  - *acknowledgement conditions 5-19*
  - *definition 5-15*
  - *enabling/disabling with INTM bit 5-19*
  - *flag bits in ICR 5-24*
  - *flag bits in IFR 5-20*
  - *flow chart of operation 5-20*
  - *flow chart of requesting $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ 5-18*
  - *interrupt mode bit (INTM) 3-16*
  - *masking/unmasking in ICR 5-24 to 5-42*
  - *masking/unmasking in IMR 5-23 to 5-42*
- nonmaskable interrupt 5-27 to 5-29
  - *definition 5-15*
  - *flow chart of operation 5-29*
  - *hardware-initiated 5-27*
  - *software-initiated 5-27*
- operation (three phases) 5-15
- pending interrupt (IFR flag set) 5-20 to 5-22
- phases of operation 5-15
- priorities
  - *'C203/C204 5-16*
  - *'C209 11-10*
  - *in interrupt acknowledgement process 5-19*
- registers
  - *interrupt control register (ICR) 5-24*
  - *interrupt flag register (IFR) 5-20 to 5-22*
    - 'C209 11-12
  - *interrupt mask register (IMR) 5-23 to 5-24*
    - 'C209 11-13

interrupt (continued)
software interrupt
definition 5-15
instructions 5-27
special cases
clearing ICR flag bits 5-25
clearing IFR flag bit after INTR instruction 5-21
clearing IFR flag bits set by serial port interrupts 5-21
controlling $\overline{INT2}$ and $\overline{INT3}$ with ICR 5-25
requesting $\overline{INT2}$ and $\overline{INT3}$ 5-18
table 5-16
vector locations
'C203/C204 5-16
'C209 11-10

interrupt acknowledge signal ($\overline{IACK}$) 11-13

interrupt control register (ICR) 5-24 to 5-42
bits 5-26
quick reference A-8

interrupt flag register (IFR) 5-20 to 5-42
bits
'C203/C204 5-21
'C209 11-12
clearing interrupts 5-20
quick reference A-6

interrupt latency
definition G-12
description 5-31

interrupt mask register (IMR) 5-23 to 5-42
bits
'C203/C204 5-23
'C209 11-13
in interrupt acknowledgement process 5-19
quick reference A-7

interrupt mode bit (INTM) 3-16

interrupt phases of operation 5-15

interrupt service routines (ISRs) 5-29
definition G-12
ISRs within ISRs 5-30
saving and restoring context 5-29

INTM (interrupt mode bit) 3-16
effect on power-down mode 5-40
in interrupt acknowledgement process 5-19

INTR instruction 7-71
introduction 5-27
operand (K) values
'C203/C204 5-16
'C209 11-10

introduction
TMS320 devices 1-2
TMS320C2xx devices 1-4

IO0–IO3 (bits) 10-13
reading current logic level on pins IO0–IO3 10-17

IO0–IO3 (pins) 10-15 to 10-17

IOSR (I/O status register)
detecting change on pins IO0–IO3 10-17
quick reference A-18
reading current logic level on pins IO0–IO3 10-17

IR (instruction register)
definition G-11

$\overline{IS}$ (I/O space select pin)
definition 4-3
shown in figure 4-17

ISR (interrupt service routine) 5-29 to 5-30
definition G-12
ISRs within ISRs 5-30
saving and restoring context 5-29 to 5-30

ISWS bit(s)
'C203/C204 8-16
'C209 11-18

J

JTAG F-16

JTAG emulator
buffered signals F-10
connection to target system F-1 to F-25
no signal buffering F-10

K

key features of the 'C2xx 1-5

L

LACC instruction 7-74

LACL instruction 7-75

LACT instruction 7-78

LAR instruction 7-80

latch phase of CPU cycle G-13

latency, interrupt 5-31 to 5-32
after execution of RET 5-32
during execution of CLRC INTM 5-32
minimum latency 5-31

LDP instruction 7-83
local data memory
  address map
    'C203 4-23
    'C209 11-6
  configuration
    RAM (dual-access)
      'C203 4-24
      'C209 11-8
    RAM (single-access) 11-7
  description 4-7
  external interfacing 4-9
    caution about proper timing 4-9
  pages of (diagram) 4-7
logic instructions
  AND 7-34
  CMPL (complement/NOT) 7-64
  OR 7-129
  XOR (exclusive OR) 7-193
logic phase of CPU cycle G-13
long immediate addressing 6-2
LPH instruction 7-85
LST instruction 7-87
LT instruction 7-91
LTA instruction 7-93
LTD instruction 7-95
LTP instruction 7-98
LTS instruction 7-100

M

MAC instruction 7-102
MACD instruction 7-106
MAR instruction 7-111
mask bits
  asynchronous serial port control register (ASPCR) 10-8
  interrupt control register (ICR) 5-24
  interrupt mask register (IMR) 5-23
maskable interrupts 5-18
  acknowledgement conditions 5-19
  definition 5-15
  enabling/disabling with INTM bit 5-19
  flag bits in ICR 5-24
  flag bits in IFR 5-20
  flow chart of operation 5-20
  flow chart of requesting $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ 5-18
maskable interrupts (continued)
  masking/unmasking in ICR 5-24
  masking/unmasking in IMR 5-23
MCM bit 9-11
memory
  address map
    'C203 4-23
    'C209 11-6
    data page 0 4-8
  available on TMS320C2xx devices 2-7
  available types 1-5
  bootloader 4-30
    boot source (EPROM) 4-30
    diagram 4-30
    enabling 4-33
    execution 4-34
    generating code for EPROM D-23 to D-24
    program code 4-37
  data page pointer (DP) 3-16
  device-specific information 4-22
  direct memory access (using HOLD operation) 4-18
    during reset 4-20
    example 4-19
    terminating correctly 4-20
  external interfacing
    global data memory 4-13
    I/O ports 4-16
    local data memory 4-9
    program memory 4-5
  flash, introduction 2-9
  global data memory 4-11 to 4-13
  HOLD operation 4-18 to 4-21
    during reset 4-20
    example 4-19
    terminating correctly 4-20
  introduction 4-2
  local data memory
    description 4-7 to 4-10
    pages of (diagram) 4-7
  on-chip memory, advantages 4-2
  organization 4-2
  overview 2-7
  pins for external interfacing 4-3
  program memory 4-5 to 4-6
    address generation logic 5-2
    address sources 5-3

memory (continued)
RAM (dual-access)
configuration
'C203 4-24
'C209 11-8
description 2-7
RAM (single-access)
configuration 11-7
description 2-8
reset conditions 5-35
ROM
configuration, 'C209 11-7
introduction 2-8

memory instructions
block move from data memory to data memory (BLDD) 7-49
block move from program memory to data memory (BLPD) 7-54
move data after add PREG to accumulator, load TREG, and multiply (MACD) 7-106
move data to next higher address in data memory (DMOV) 7-66
move data, load TREG, and add PREG to accumulator (LTD) 7-95
store long immediate value to data memory (SPLK) 7-165
table read (TBLR) 7-186
table write (TBLW) 7-189
transfer data from data memory to I/O space (OUT) 7-132
transfer data from I/O space to data memory (IN) 7-69
transfer word from data memory to program memory (TBLW) 7-189
transfer word from program memory to data memory (TBLR) 7-186

memory-mapped registers
addresses and reset values A-2

micro stack (MSTACK) 5-6

microprocessor/microcomputer pin (MP/$\overline{MC}$)
definition 4-4
use in configuring memory, 'C209 11-7

MINT2 bit 5-27

MINT3 bit 5-27

MODE bit 5-26
used in HOLD operation 4-18

MP/$\overline{MC}$ (microprocessor/microcomputer pin)
definition 4-4
use in configuring memory, 'C209 11-7

MPY instruction 7-113

MPYA instruction 7-116

MPYS instruction 7-118

MPYU instruction 7-120

MSTACK (micro stack) 5-6

multicycle instructions 5-31

multiplication section of CPU 3-5

multiplier
description 3-5
introduction 2-6

multiply instructions
multiply (include load to TREG) and accumulate previous product (MAC) 7-102
multiply (include load to TREG), accumulate previous product, and move data (MACD) 7-106
multiply (MPY) 7-113
multiply and accumulate previous product (MPYA) 7-116
multiply and subtract previous product (MPYS) 7-118
multiply unsigned (MPYU) 7-120
square specified value after accumulating previous product (SQRA) 7-168
square specified value after subtracting previous product from accumulator (SQRS) 7-170

N

NEG instruction 7-122

next auxiliary register 6-11

next program address register (NPAR)
definition G-14
shown in figure 5-2

$\overline{NMI}$ hardware interrupt
description 5-27
priority
'C203/C204 5-17
'C209 11-11
vector location
'C203/C204 5-17
'C209 11-11

$\overline{NMI}$ instruction 7-124
introduction 5-28
vector location
'C203/C204 5-17
'C209 11-11

nonmaskable interrupts 5-27
- definition 5-15
- flow chart of operation 5-29
- hardware-initiated 5-27
- software-initiated 5-27

NOP instruction 7-125

NORM instruction 7-126

NPAR (next program address register)
- definition G-14
- shown in figure 5-2

O

OE bit 10-11

off-chip (external) memory
- ’C203 4-23
- ’C209 11-6

on-chip generators
- baud-rate generator 10-4
- clock generator 8-4
  - *’C209 clock options 11-15*
- wait-state generator 8-15
  - *’C209 11-17*

on-chip memory
- advantages 4-2
- flash, introduction 2-9
- RAM (dual-access)
  - *available*
    - ’C203 4-23
    - ’C209 11-6
  - *configuration*
    - ’C203 4-24
    - ’C209 11-8
  - *description 2-7*
- RAM (single-access)
  - *available, ’C209 11-6*
  - *configuration 11-7*
  - *description 2-8*
- ROM
  - *available, ’C209 11-6*
  - *configuration, ’C209 11-7*
  - *introduction 2-8*

on-chip peripherals
- asynchronous serial port 10-1 to 10-20
- available types 1-6
- clock generator 8-4 to 8-6
  - *’C209 clock options 11-15 to 11-18*
- control of 8-2 to 8-3
- general-purpose I/O pins 8-18 to 8-21

on-chip peripherals (continued)
- overview 2-11
- register locations and reset values 8-2
- reset conditions 5-36, 8-2
- synchronous serial port 9-1 to 9-42
- timer 8-8 to 8-14
- wait-state generator 8-15 to 8-17
  - *’C209 11-17 to 11-18*

on-chip registers mapped to data space
- addresses and reset values A-2
- quick reference figures A-4

on-chip registers mapped to I/O space
- addresses and reset values A-2
- quick reference figures A-4

on-chip ROM E-1

opcode format
- direct addressing 6-5
- immediate addressing 6-2
- indirect addressing 6-12

OR instruction 7-129

oscillator 8-4

OUT instruction 7-132

output modes
- external count F-20
- signal event F-20

output shifter 3-11

OV (overflow flag bit) 3-16

overflow in accumulator
- detecting (OV bit) 3-16
- enabling/disabling overflow mode (OVM bit) 3-17

overflow in synchronous serial port
- burst mode 9-27
- continuous mode 9-28
- detecting (OVF bit) 9-10

overflow mode bit (OVM) 3-17
- effects on accumulator 3-10

OVF bit 9-10

P

PAB (program address bus)
- definition 2-3
- used in program-memory address generation 5-3

PAC instruction 7-134

packages
- available types 1-6

pages of data memory
  figure 6-4
PAL F-21, F-22, F-24
PAR (program address register)
  definition G-16
  shown in figure 5-2
parallel I/O ports 4-14
PC (program counter) 5-3
  description 5-3
  loading 5-4
  shown in figure 5-2
peripherals (on-chip)
  asynchronous serial port 10-1 to 10-20
  available types 1-6
  clock generator 8-4 to 8-6
    'C209 clock options 11-15 to 11-18
  control of 8-2 to 8-3
  general-purpose I/O pins 8-18 to 8-21
  overview 2-11
  register locations and reset values 8-2
  reset conditions 5-36, 8-2
  synchronous serial port 9-1 to 9-42
  timer 8-8 to 8-14
  wait-state generator 8-15 to 8-17
    'C209 11-17 to 11-18
phase lock loop (PLL) 8-4
pins
  asynchronous serial port 10-4
  CLKOUT1 8-7
  clock generator
    CLKIN/X2 8-4
    CLKMOD 11-15
    DIV1 and DIV2 8-5
    X1 8-4
  general-purpose
    $\overline{\text{BIO}}$ 8-18
    IO0–IO3 10-15
    XF 8-19
  I/O and memory 4-3
  $\overline{\text{IACK}}$ ('C209) 11-13
  memory and I/O 4-3
  READY 8-15
  synchronous serial port 9-4
  timer (TOUT) 8-8
  wait-state (READY) 8-15
pipeline
  operation 5-7
PM (product shift mode bits) 3-17
POP instruction 7-135
pop operation (diagram) 5-6
POPD instruction 7-137
power saving features 1-6
power-down mode 5-40
PRD G-22
PRD (timer period register) 8-12 to 8-13, G-22 to G-26
PRDB (program read bus) 2-3
PREG (product register) 3-6
PREG instructions
  add PREG to accumulator (APAC) 7-37
  add PREG to accumulator and load TREG (LTA) 7-93
  add PREG to accumulator and multiply (MPYA) 7-116
  add PREG to accumulator and square specified value (SQRA) 7-168
  add PREG to accumulator, load TREG, and move data (LTD) 7-95
  add PREG to accumulator, load TREG, and multiply (MAC) 7-102
  add PREG to accumulator, load TREG, multiply, and move data (MACD) 7-106
  load high bits of PREG (LPH) 7-85
  set PREG output shift mode (SPM) 7-167
  store high word of PREG to data memory (SPH) 7-161
  store low word of PREG to data memory (SPL) 7-163
  store PREG to accumulator (PAC instruction) 7-134
  store PREG to accumulator and load TREG (LTP) 7-98
  subtract PREG from accumulator (SPAC) 7-160
  subtract PREG from accumulator and load TREG (LTS) 7-100
  subtract PREG from accumulator and multiply (MPYS) 7-118
  subtract PREG from accumulator and square specified value (SQRS) 7-170
product register (PREG) 3-6
product shift mode bits (PM) 3-17
product shift modes 3-7
product shifter 3-6
program address bus (PAB)
  definition 2-3
  used in program-memory address generation 5-3

program address register (PAR)
  definition G-16
  shown in figure 5-2

program control features
  address generation, program memory 5-2
  branch instructions
    conditional 5-11
    unconditional 5-8
  call instructions
    conditional 5-12
    unconditional 5-8
  conditional instructions 5-10 to 5-13
    conditions that may be tested 5-10 to 5-13
    stabilization of conditions 5-11 to 5-13
    using multiple conditions 5-10
  pipeline operation 5-7
  program counter (PC) 5-3
    loading 5-4
  repeating a single instruction 5-14
  reset conditions 5-35
  return instructions
    conditional 5-12
    unconditional 5-9
  stack 5-4
  status registers ST0 and ST1 3-15
    bits 3-15

program counter (PC) 5-3
  description 5-3
  loading 5-4
  shown in figure 5-2

program examples D-1 to D-24
  about the examples D-2
  asynchronous serial port
    automatic baud-rate detection test D-16
    delta interrupts D-18
    transmission D-13
    transmission loopback test D-14
  boot loader code
    command file D-24
    hex conversion file D-24
  command file (generic) D-5
  delay loops D-8
  header file with I/O register declarations D-6
  header file with interrupt vector declarations D-7
  HOLD operation D-11
  interrupt $\overline{\text{INT1}}$ D-10
  interrupts $\overline{\text{INT2}}$ and $\overline{\text{INT3}}$ D-12

program examples (continued)
  synchronous serial port
    transmission (continuous mode) D-20
    using with codec D-21
  timer D-9

program memory
  address generation logic 5-2
    micro stack (MSTACK) 5-6
    program counter (PC) 5-3
    stack 5-4
  address map
    'C203 4-23
    'C209 11-6
  address sources 5-3
  configuration
    RAM (dual-access)
      'C203 4-24
      'C209 11-8
    RAM (single-access) 11-7
    ROM, 'C209 11-7
  description 4-5
  external interfacing 4-5
    caution about proper timing 4-5

program memory select pin (PS)
  definition 4-3
  shown in figure 4-6

program read bus (PRDB) 2-3

program-address generation (diagram) 5-2

protocol
  bus, in emulator system F-4

$\overline{\text{PS}}$ (program memory select pin)
  definition 4-3
  shown in figure 4-6

PSC (timer prescaler counter)
  'C203/C204 8-11
  'C209 11-16
  definition G-17

PSHD instruction 7-139

PSLWS bits 8-16

PSUWS bits 8-16

PSWS bit 11-18

PUSH instruction 7-141

push operation (diagram) 5-5

R

R/$\overline{W}$ (read/write pin) 4-4

RAM (on-chip)
- dual-access
  - *configuration*
    - 'C203 4-24
    - 'C209 11-8
  - *description 2-7*
- single-access
  - *configuration 11-7*
  - *description 2-8*

RAMEN (single-access RAM enable pin)
- definition 4-4
- use in configuring memory 11-7

$\overline{RD}$ (read select pin)
- definition 4-4
- shown in figure 4-6, 4-10, 4-13, 4-31

read select pin ($\overline{RD}$)
- definition 4-4
- shown in figure 4-6, 4-10, 4-13, 4-31

read/write pin (R/$\overline{W}$) 4-4

READY (external device ready pin)
- definition 4-4
- generating wait states with 8-15

receive interrupt
- asynchronous serial port 10-17
  - *enabling/disabling (RIM bit) 10-8*
- synchronous serial port 9-6

receive pin
- asynchronous serial port (RX) 10-4
  - *detecting break on (BI bit) 10-10*
- synchronous serial port (DR) 9-4

receive register
- asynchronous serial port (ADTR) 10-4
  - *detecting overrun in (OE bit) 10-11*
  - *detecting when empty (DR bit) 10-12*
- synchronous serial port (SDTR) 9-5

receive shift register
- asynchronous serial port (ARSR) 10-5
- synchronous serial port (RSR) 9-5

register summary A-1 to A-18

registers
- addresses and reset values A-2
- asynchronous serial port
  - *baud-rate divisor register (BRD) 10-14*
  - *control register (ASPCR) 10-7*
  - *I/O status register (IOSR) 10-10*
  - *receive shift register (ARSR) 10-5*
  - *transmit shift register (AXSR) 10-5*
- auxiliary registers, current auxiliary register 6-13
- auxiliary registers (AR0–AR7)
  - *current auxiliary register 6-9*
  - *next auxiliary register 6-11*
- baud-rate divisor register (BRD) 10-14
- CLKOUT1-pin control (CLK) register 8-7
- I/O status register (IOSR) 10-10
- interrupt control register (ICR) 5-24 to 5-42
- interrupt flag register (IFR) 5-20 to 5-22
  - *'C209 11-12 to 11-18*
- interrupt mask register (IMR) 5-23 to 5-24
  - *'C209 11-13 to 11-18*
- mapped to data page 0 4-8
- mapped to I/O space
  - *'C203/C204 4-16*
  - *'C209 11-9*
  - *accessing 4-16*
- quick reference A-1 to A-18
- serial port 9-32
- status registers ST0 and ST1 3-15
- timer
  - *control register (TCR)*
    - 'C203/C204 8-10
    - 'C209 11-17
  - *counter register (TIM) 8-12, G-22*
  - *divide-down register (TDDR)*
    - 'C203/C204 8-12
    - 'C209 11-17
  - *period register (PRD) 8-12, G-22*
  - *prescaler counter (PSC)*
    - 'C203/C204 8-11
    - 'C209 11-16
- wait-state generator control register (WSGR)
  - *'C203/C204 8-16*
  - *'C209 11-18*

repeat (RPT) instruction
- description 7-146
- introduction 5-14

repeat counter (RPTC) 5-14

repeating a single instruction 5-14

reset 5-35
- at same time as HOLD operation 4-20
- introduction 5-27
- priority
  - *'C203/C204 5-16*
  - *'C209 11-10*
- vector location
  - *'C203/C204 5-16*
  - *'C209 11-10*

reset values of on-chip registers
- mapped to data space 5-37, A-2
- mapped to I/O space 5-37, A-2
- status registers ST0 and ST1 A-2

RET instruction 7-142

RETC instruction 7-143

return instructions
- conditional, overview 5-12
- return conditionally from subroutine (RETC) 7-143
- return unconditionally from subroutine (RET) 7-142
- unconditional, overview 5-9

RFNE bit 9-9

RIM bit 10-8

RINT bit
- in interrupt flag register (IFR) 5-22
- in interrupt mask register (IMR) 5-23

RINT interrupt
- definition G-19
- flag bit 5-22
- mask bit 5-23
- priority 5-16
- vector location 5-16

ROL instruction 7-144

ROM, customized E-1 to E-3

ROM (on-chip)
- configuration, 'C209 11-7
- introduction 2-8

ROM codes
- submitting to Texas Instruments E-1 to E-3

ROR instruction 7-145

RPT instruction 7-146

RPTC (repeat counter) 5-14

RRST bit 9-10

$\overline{\text{RS}}$ (reset)
- at same time as HOLD operation 4-20
- introduction 5-27
- priority
  - *'C203/C204 5-16*
  - *'C209 11-10*
- vector location
  - *'C203/C204 5-16*
  - *'C209 11-10*

RSR (synchronous serial port receive shift register) 9-5

run/stop operation F-10

RUNB
- debugger command F-20 to F-24

RUNB_ENABLE
- input F-22

RX pin 10-4

S

SACH instruction 7-148

SACL instruction 7-150

SAR instruction 7-152

SARAM (single-access RAM)
- configuration 11-7
- definition G-19
- description 2-8

SBRK instruction 7-154

scaling shifters
- input shifter 3-3
- introduction 2-5
- output shifter 3-11
- product shifter 3-6
  - *product shift modes 3-7*

scan path linkers F-16
- secondary JTAG scan chain to an SPL F-17
- suggested timings F-22
- usage F-16

scan paths
- TBC emulation connections for JTAG scan paths F-25

scanning logic overview 2-13

SDTR (synchronous serial port transmit and receive register) 9-5
- using to access FIFO buffers 9-15

serial port
- registers 9-32

serial ports
- available on TMS320C2xx devices 2-12
- introduction 2-12
- reset conditions 5-36

serial-scan emulation capability 2-13

SETBRK bit 10-9

SETC instruction 7-155

SFL instruction 7-157

SFR instruction 7-158

shifters
- input shifter 3-3
- introduction 2-5
- output shifter 3-11
- product shifter 3-6
  - *product shift modes 3-7*

short immediate addressing 6-2

signal descriptions
- 14-pin header F-3

signals
- buffered F-10
- buffering for emulator connections F-10 to F-13
- description, 14-pin header F-3
- timing F-6

sign-extension mode bit (SXM)
- definition 3-17
- effect on CALU (central arithmetic logic unit) 3-9
- effect on input shifter 3-4

single-access RAM (SARAM)
- configuration 11-7
- definition G-19
- description 2-8

single-access RAM enable pin (RAMEN)
- definition 4-4
- use in configuring memory 11-7

slave devices F-4

SOFT bit 9-9
- asynchronous serial port 10-7
- timer 8-11

software interrupts
- definition 5-15
- instructions 5-27

SPAC instruction 7-160

SPH instruction 7-161

SPL instruction 7-163

SPLK instruction 7-165

SPM instruction 7-167

SQRA instruction 7-168

SQRS instruction 7-170

SSPCR (synchronous serial port control register) 9-8
- quick reference A-16

SST instruction 7-172

stack 5-4
- managing nested interrupt service routines 5-30
- pop top of stack to data memory (POPD instruction) 7-137
- pop top of stack to low accumulator bits (POP instruction) 7-135
- push data memory value onto stack (PSHD instruction) 7-139
- push low accumulator bits onto stack (PUSH instruction) 7-141

status registers ST0 and ST1
- addresses and reset values A-2
- bits 3-15
- clear control bit (CLRC instruction) 7-62
- introduction 3-15
- load (LST instruction) 7-87
- load data page pointer (LDP instruction) 7-83
- modify auxiliary register pointer (MAR instruction) 7-111
- quick reference A-5
- set control bit (SETC instruction) 7-155
- set product shift mode (SPM instruction) 7-167
- store (SST instruction) 7-172

STB bit 10-8

$\overline{\text{STRB}}$ (external access active strobe) 4-3

SUB instruction 7-174

SUBB instruction 7-178

SUBC instruction 7-180

SUBS instruction 7-182

SUBT instruction 7-184

SXM (sign-extension mode bit)
- definition 3-17
- effect on CALU (central arithmetic logic unit) 3-9
- effect on input shifter 3-4

synchronous serial port
- basic operation 9-6
- bit input from CLKR pin (IN0 bit) 9-10
- block diagram 9-3
- burst mode (introduction) 9-12
- CLKR pin as bit input (IN0 bit) 9-10
- clock source for transmission (MCM bit) 9-12
- components 9-3

synchronous serial port (continued)
- configuration 9-8
- continuous mode (introduction) 9-12
- controlling and resetting 9-8
- digital loopback mode 9-26
- emulation modes 9-26
- error conditions
  - *burst mode 9-27*
  - *continuous mode 9-27*
- features 9-1
- FIFO buffers
  - *detecting data in receive FIFO buffer (RFNE bit) 9-9*
  - *detecting empty transmit FIFO buffer (TCOMP bit) 9-9*
  - *introduction 9-5*
  - *managing contents with SDTR 9-15*
- frame sync modes (FSM bit) 9-12
- frame sync source for transmission (TXM bit) 9-12
- interrupts (XINT and RINT)
  - *flag bits 5-22*
  - *mask bits 5-23*
  - *priorities 5-16*
  - *receive (RINT) 9-6*
    - controlling (FR1 and FR0 bits) 9-10
  - *transmit (XINT) 9-6*
    - controlling (FT1 and FT0 bits) 9-9
  - *using 9-13*
  - *vector locations 5-16*
- introduction 2-12
- overflow in receiver
  - *burst mode 9-27*
  - *continuous mode 9-28*
  - *detecting (OVF bit) 9-10*
- overview 9-2
- pins 9-4
- receiver operation 9-22
  - *burst mode 9-22*
  - *continuous mode 9-23*
- registers (overview) 9-5
- reset conditions 5-36
- resetting 9-13
  - *receiver (RRST bit) 9-10*
  - *transmitter (XRST bit) 9-10*
- selecting mode of operation 9-12
- selecting transmit clock source 9-12
- selecting transmit frame sync source 9-12
- signals 9-3
- testing 9-25

synchronous serial port (continued)
- transmitter operation 9-16
  - *burst mode with external frame sync 9-17*
  - *burst mode with internal frame sync 9-16*
  - *continuous mode with external frame sync 9-20*
  - *continuous mode with internal frame sync 9-19*
- troubleshooting
  - *bits for testing the port 9-25*
  - *error conditions*
    - burst mode 9-27
    - continuous mode 9-27
- underflow in transmitter
  - *burst mode 9-27*
  - *continuous mode 9-27*

synchronous serial port registers
- control register (SSPCR)
  - *description 9-8*
  - *quick reference A-16*
- FIFO buffers
  - *detecting data in receive FIFO buffer (RFNE bit) 9-9*
  - *detecting empty transmit FIFO buffer (TCOMP bit) 9-9*
  - *introduction 9-5*
  - *managing contents with SDTR 9-15*
- overview 9-5
- receive shift register (RSR) 9-5
- transmit and receive register (SDTR) 9-5
  - *using to access FIFO buffers 9-15*
- transmit shift register (XSR) 9-5

T

target cable F-14

target system
- connection to emulator F-1 to F-25

target system emulator connector
- designing F-2

target-system clock F-12

TBLR instruction 7-186

TBLW instruction 7-189

TC (test/control flag bit) 3-17
- response to accumulator event 3-10
- response to auxiliary register compare 3-14

TCK signal F-2, F-3, F-4, F-6, F-7, F-13, F-17, F-18, F-25

TCOMP bit 9-9

TCR (timer control register) 8-10 to 8-12
- 'C209 11-16
- quick reference A-9

TDDR (timer divide-down register)
- 'C203/C204 8-12
- 'C209 11-17
- definition G-22

TDI signal F-2 to F-8, F-13, F-18

TDO signal F-4, F-5, F-8, F-19, F-25

temporary register (TREG) 3-6

TEMT bit 10-11

test bus controller F-22, F-24

test clock F-12
- diagram F-12

test/control flag bit (TC) 3-17
- response to accumulator event 3-10
- response to auxiliary register compare 3-14

THRE bit 10-11

TIM (timer counter register) 8-12 to 8-13, G-22 to G-26

TIM bit 10-8

timer 8-8 to 8-14
- block diagram 8-8
- control register (TCR) 8-10 to 8-12
- counter register (TIM) 8-12 to 8-13, G-22 to G-26
- divide-down register (TDDR)
  - *'C203/C204 8-12*
  - *'C209 11-17*
  - *definition G-22*
- interrupt (TINT)
  - *'C203/C204*
    - flag bit 5-22
    - mask bit 5-24
    - priority 5-16
    - vector location 5-16
  - *'C209*
    - flag bit 11-12
    - mask bit 11-13
    - priority 11-10
    - vector location 11-10
- interrupt rate 8-13
- operation 8-9 to 8-10
- period register (PRD) 8-12 to 8-13, G-22 to G-26
- prescaler counter (PSC)
  - *'C203/C204 8-11*
  - *'C209 11-16*
- reload
  - *'C203/C204 8-11*
  - *'C209 11-17*
- reset 8-14
- setting interrupt rate 8-13
- stop/start
  - *'C203/C204 8-12*
  - *'C209 11-17*

timer control register (TCR) 8-10 to 8-12
- 'C209 11-16
- quick reference A-9

timer counter register (TIM) 8-12 to 8-13, G-22 to G-26

timer period register (PRD) 8-12 to 8-13, G-22 to G-26

timing calculations F-7 to F-9, F-18 to F-26

TINT bit
- 'C203/C204
  - *in interrupt flag register (IFR) 5-22*
  - *in interrupt mask register (IMR) 5-24*
- 'C209
  - *in interrupt flag register (IFR) 11-12*
  - *in interrupt mask register (IMR) 11-13*

TINT interrupt
- 'C203/C204
  - *flag bit 5-22*
  - *mask bit 5-24*
  - *priority 5-16*
  - *vector location 5-16*
- 'C209
  - *flag bit 11-12*
  - *mask bit 11-13*
  - *priority 11-10*
  - *vector location 11-10*
- definition G-22

TMS signal F-2 to F-8, F-13, F-17 to F-19, F-25

TMS/TDI inputs F-4

TMS320 devices
- applications 1-3
- overview 1-2

TMS320 ROM code submittal
- flow chart E-2

TMS320C1x/C2x/C2xx/C5x instruction set
- comparisons B-1 to B-10, C-1 to C-36

TMS320C209 device 11-1 to 11-18
- comparison to other 'C2xx devices 11-2
  - *differences in interrupts 11-3*
  - *differences in memory and I/O spaces 11-3*
  - *differences in peripherals 11-2*
  - *similarities 11-2*

TMS320C209 device (continued)
- interrupts 11-10
- locating ’C209 information in this manual (table) 11-3
- memory and I/O spaces 11-5
- on-chip peripherals 11-15

transmit interrupt
- asynchronous serial port 10-17
  - *enabling/disabling (TIM bit) 10-8*
- synchronous serial port 9-6

transmit pin
- asynchronous serial port (TX) 10-4
  - *output level between transmissions (SETBRK bit) 10-9*
- synchronous serial port (DX) 9-4

transmit register
- asynchronous serial port (ADTR) 10-4
  - *detecting when empty (THRE bit) 10-11*
  - *detecting when it and AXSR are empty (TEMT bit) 10-11*
- synchronous serial port (SDTR) 9-5

transmit shift register
- asynchronous serial port (AXSR) 10-5
  - *detecting when it and ADTR are empty (TEMT bit) 10-11*
- synchronous serial port (XSR) 9-5

TRAP instruction 7-192
- introduction 5-28
- vector location
  - *’C203/C204 5-17*
  - *’C209 11-11*

TRB bit
- ’C203/C204 8-11
- ’C209 11-17

TREG (temporary register) 3-6

TREG instructions
- load accumulator using shift specified by TREG (LACT) 7-78
- load TREG (LT) 7-91
- load TREG and add PREG to accumulator (LTA) 7-93
- load TREG and store PREG to accumulator (LTP) 7-98
- load TREG and subtract PREG from accumulator (LTS) 7-100
- load TREG, add PREG to accumulator, and move data (LTD) 7-95

TREG instructions (continued)
- load TREG, add PREG to accumulator, and multiply (MAC) 7-102
- load TREG, add PREG to accumulator, multiply, and move data (MACD) 7-106

$\overline{\text{TRST}}$ signal F-2, F-3, F-6, F-7, F-13, F-17, F-18, F-25

TSS bit
- ’C203/C204 8-12
- ’C209 11-17

TX pin 10-4

TXM bit 9-10

TXRXINT bit
- in interrupt flag register (IFR) 5-21
- in interrupt mask register (IMR) 5-23

TXRXINT interrupt
- flag bit 5-21
- mask bit in IMR 5-23
- priority 5-16
- vector location 5-16

U

unconditional instructions
- unconditional branch 5-8
- unconditional call 5-8
- unconditional return 5-9

underflow in synchronous serial port
- burst mode 9-27
- continuous mode 9-27

URST bit 10-8

W

wait states
- definition G-24
- for data space
  - *’C203/C204 8-16*
  - *’C209 11-18*
- for I/O space
  - *’C203/C204 8-16*
  - *’C209 11-18*
- for program space
  - *’C203/C204 8-16*
  - *’C209 11-18*
- generating with READY signal 8-15
- generating with wait-state generator
  - *’C203/C204 8-15 to 8-18*
  - *’C209 11-17 to 11-18*

wait-state generator 8-15 to 8-17
’C209 11-17 to 11-18
introduction 2-11
wait-state generator control register (WSGR) 8-16
’C209 11-18
quick reference A-10
$\overline{WE}$ (write enable pin)
definition 4-4
shown in figure 4-6, 4-10, 4-13, 4-17
write enable pin ($\overline{WE}$)
definition 4-4
shown in figure 4-6, 4-10, 4-13, 4-17
WSGR (wait-state generator control register)
’C203/C204 8-16
’C209 11-18
quick reference A-10

X

XF bit (XF pin status bit) 3-17
XF pin 8-19
XINT bit
in interrupt flag register (IFR) 5-22
in interrupt mask register (IMR) 5-23
XINT interrupt
flag bit 5-22
mask bit 5-23
priority 5-16
vector location 5-16
XOR instruction 7-193
XRST bit 9-10
XSR (synchronous serial port transmit shift register) 9-5

Z

ZALR instruction 7-196