TMS320C28x DSP CPU and Instruction Set Reference Guide

Literature Number: SPRU430D August 2001 – Revised March 2004



IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| Products | | Applications | |
|------------------|------------------------|--------------------|---------------------------|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| | | Telephony | www.ti.com/telephony |
| | | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |

Mailing Address:

Texas Instruments

Post Office Box 655303 Dallas, Texas 75265

Copyright © 2004, Texas Instruments Incorporated

Read This First

About This Manual

This manual describes the central processing unit (CPU) and the assembly language instructions of the TMS320C28x 32-bit fixed-point digital signal processors (DSPs). It also describes emulation features available on these DSPs. A summary of the chapters and appendixes follows:

Chapter 1 Architectural Overview

This chapter introduces the T320C2800 DSP core that is at the heart of each TMS320C28x DSP. The chapter includes a memory map and a high-level description of the memory interface that connects the core with memory and peripheral devices.

Chapter 2 Central Processing Unit

This chapter describes the architecture, registers, and primary functions of the CPU. The chapter includes detailed descriptions of the flag and control bits in the most important CPU registers, status registers ST0 and ST1.

Chapter 3 Interrupts and Reset

This chapter describes the interrupts and how they are handled by the CPU. The chapter also explains the effects of a reset on the CPU and includes discussion of the automatic context save performed by the CPU prior to servicing an interrupt.

Chapter 4 Pipeline

This chapter describes the phases and operation of the instruction pipeline. The chapter is primarily for readers interested in increasing the efficiency of their programs by preventing pipeline delays.

Chapter 5 Addressing Modes

This chapter explains the modes by which the assembly language instructions accept data and access register and memory locations. The chapter includes a description of how addressing-mode information is encoded in opcodes.

Chapter 6 Assembly Language Instructions

This chapter provides summaries of the instruction set and detailed descriptions (including examples) for the instructions. The chapter includes an explanation of how 32-bit accesses are aligned to even addresses.

| Chapter 7 | Emulation Features This chapter describes the TMS320C28x emulation features that can be used with only a JTAG port and two additional emulation pins. |
|------------|--|
| Appendix A | Register Quick Reference This appendix is a concise central resource for information about the status and control registers of the CPU. The chapter includes figures that summa- rize the bit fields of the registers. |
| Appendix B | Submitting ROM Codes to TI This appendix describes the procedures for getting code-customized ROM in a Texas Instruments (TI [™]) DSP. |
| Appendix C | C2xLP and C28x Architectural Differences This appendix describes the differences in the architecture of the C2xLP and the C28x. |
| Appendix D | Migration From C2xLP This appendix explains how to migrate code from the C2xLP to the C28x. |
| Appendix E | C2xLP Instruction Set Compatibility This appendix describes the instruction set compatibility with the C2xLP. |
| Appendix F | Migration From C27x to C28x This appendix explains how to migrate code from the C27x to the C28x. |
| Appendix G | Glossary This appendix explains abbreviations, acronyms, and special terminology used throughout this document. |
| | |

Notational Conventions

This document uses the following conventions:

- The device number TMS320C28x is very often abbreviated as '28x.
- □ Program examples are shown in a special typeface. Here is a sample line of program code:

PUSH IER

Portions of an instruction syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* are variables indicating information that should be entered. Here is an example of an instruction syntax:

MOV ARx, *-SP[6bit]

MOV is the instruction mnemonic. This instruction has two operands, indicated by **AR***x* and *****–**SP**[6bit]. Where the variable *x* appears, you type a

value from 0 to 5; where the *6bit* appears, you type a 6-bit constant. The rest of the instruction, including the square brackets, must be entered as shown.

❑ When braces or brackets enclose an operand, as in {operand}, the operand is optional. If you use an optional operand, you specify the information within the braces; you do not enter the braces themselves. In the following syntax, the operand << *shift* is optional:

```
MOV ACC, *-SP[6bit] {<< shift }
```

MOV ACC, *-**SP**{*6bit*} {<< *shift* }

For example, you could use either of the following instructions:

MOV ACC, *-SP[5] MOV ACC, *-SP[5]<< 4

In most cases, hexadecimal numbers are shown with a subscript of 16. For example, the hexadecimal number 40 would be shown as 40₁₆. An exception to this rule is a hexadecimal number in a code example; these hexadecimal numbers have the suffix h. For example, the number 40 in the following code is a hexadecimal 40.

MOVB AR0,#40h

Similarly, binary numbers usually are shown with a subscript of 2. For example, the binary number 4 would be shown as 0100₂. Binary numbers in example code have the suffix b. For example, the following code uses a binary 4.

MOVB AR0,#0100b

Bus signals and bits are sometimes represented with the following notations:

| Notation | Description | Example |
|---------------|------------------------------|---|
| Bus(n:m) | Signals n through m of bus | PRDB(31:0) represents the 32 signals of the program-read data bus (PRDB). |
| Register(n:m) | Bits n through m of register | T(3:0) represents the 4 least sig- nificant bits of the T register. |
| Register(n) | Bit n of register | IER(4) represents bit 4 of the in- terrupt enable register (IER). |

v

| Notation | Description | Example |
|----------|-----------------------|---|
| x:y | x concatenated with y | AR1:AR0 is the concatenation of the 16-bit registers AR1 and AR0. AR0 is the low word. AR1 is the high word. |

Concatenated values are represented with the following notation:

□ If a signal is from an active-low pin, the name of the signal is qualified with an overbar (for example, INT1). If a signal is from an active-high pin or from hardware inside the the DSP (in which case, the polarity is irrelevant), the name of the signal is left unqualified (for example, DLOGINT).

Related Documentation From Texas Instruments

The following books describe the TMS320C28x DSP and related support tools. The documents are available for downloading on the Texas Instruments website (www.ti.com).

- **TMS320C2xx User's Guide** (literature number SPRU127) discusses the hardware aspects of the TMS320C2xx[™] 16-bit fixed-point digital signal processors. It describes the architecture, the instruction set, and the on-chip peripherals.
- **TMS320C28x Assembly Language Tools User's Guide** (literature number SPRU513) describes the assembly language tools (assembler and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C28x[™] device.
- TMS320C28x Optimizing C Compiler User's Guide (literature number SPRU514) describes the TMS320C28x[™] C/C++ compiler. This compiler accepts ANSI standard C/C++ source code and produces TMS320[™] DSP assembly language source code for the TMS320C28x device.
- TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, and TMS320C2812 Digital Signal Processors (literature number SPRS174) data sheet contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.
- *TMS320F2801, TMS320F2806, TMS320F2808 Digital Signal Processors* (literature number SPRS230) data sheet contains the pinout, signal descriptions, as well as electrical and timing specifications for the F280x devices.

- **TMS320C2800 Digital Signal Processor** (literature number SPRS178) data sheet contains the block diagram, component descriptions, timing information, and electrical specifications for the TMP320C2800 DSP.
- **TMS320C28x** Analog-to-Digital Converter (ADC) Reference Guide (literature number SPRU060) describes the ADC module. The module is a 12-bit pipelined ADC. The analog circuits of this converter, referred to as the core in this document, include the front-end analog multiplexers (MUXs), sample-and-hold (S/H) circuits, the conversion core, voltage regulators, and other analog supporting circuits. Digital circuits, referred to as the wrapper in this document, include programmable conversion sequencer, result registers, interface to analog circuits, interface to device peripheral bus, and interface to other on-chip modules.
- **TMS320C28x Boot ROM Reference Guide** (literature number SPRU095) describes the purpose and features of the bootloader (factory-programmed boot-loading software). It also describes other contents of the device on-chip boot ROM and identifies where all of the information is located within that memory.
- **TMS320C28x Enhanced Controller Area Network (eCAN) Reference Guide** (literature number SPRU074) describes the eCAN that uses established protocol to communicate serially with other controllers in electrically noisy environments. With 32 fully configurable mailboxes and time-stamping feature, the eCAN module provides a versatile and robust serial communication interface. The eCAN module implemented in the C28x DSP is compatible with the CAN 2.0B standard (active).
- TMS320C28x Event Manager (EV) Reference Guide (literature number SPRU065) describes the EV modules that provide a broad range of functions and features that are particularly useful in motion control and motor control applications. The EV modules include general-purpose (GP) timers, full-compare/PWM units, capture units, and quadrature-encoder pulse (QEP) circuits.
- TMS320C28x External Interface (XINTF) Reference Guide (literature number SPRU067) describes the various interrupts and system control features of the 28x digital signal processors (DSPs).
- **TMS320C28x Multi-channel Buffered Serial Ports (McBSPs) Reference Guide** (literature number SPRU061) describes the McBSP) available on the C28x devices. The McBSPs allow direct interface between a DSP and other devices in a system.
- **TMS320C28x Peripheral Reference Guide** (literature number SPRU566) describes the peripheral reference guides of the 28x digital signal processors (DSPs).

- TMS320C28x Serial Communication Interface (SCI) Reference Guide (literature number SPRU051) describes the SCI that is a two-wire asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format.
- TMS320C28x Serial Peripheral Interface (SPI) Reference Guide (literature number SPRU059) describes the SPI a high-speed synchronous serial input/output (I/O) port that allows a serial bit stream of programmed length (one to sixteen bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI is used for communications between the DSP controller and external peripherals or another controller.
- *TMS320C28x System Control and Interrupts Reference Guide* (literature number SPRU078) describes the various interrupts and system control features of the 28x digital signal processors (DSPs).

Trademarks

320 Hotline On-line is a trademark of Texas Instruments Incorporated.

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

IBM and PC are trademarks of International Business Machines Corporation.

Intel is a trademark of Intel Corporation.

MS-DOS is a registered trademark of Microsoft Corporation.

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

SunOS is a trademark of Sun Microsystems, Inc.

C2xLP, C27x, C28x, TMS320C28x, and XDS510 are trademarks of Texas Instruments Incorporated.

Contents

| 1 | Arch | itectural | l Overview | . 1-1 |
|-------------------|--------|-----------|--|-------|
| | Introc | luces the | e architecture and memory map of the T320C28x DSP CPU. | |
| | 1.1 | Introdu | iction to the CPU | . 1-2 |
| | | 1.1.1 | Compatibility With Other TMS320 CPUs | . 1-2 |
| | | 1.1.2 | Switching to C28x Mode From Reset | . 1-3 |
| | 1.2 | Compo | onents of the CPU | . 1-4 |
| | | 1.2.1 | Central Processing Unit (CPU) | . 1-4 |
| | | 1.2.2 | Emulation Logic | . 1-5 |
| | | 1.2.3 | Signals | . 1-6 |
| | 1.3 | Memor | y Map | . 1-7 |
| | | 1.3.1 | On-Chip Program/Data | . 1-7 |
| | | 1.3.2 | Reserved | . 1-7 |
| | | 1.3.3 | CPU Interrupt Vectors | . 1-7 |
| | 1.4 | Memor | y Interface | . 1-9 |
| | | 1.4.1 | Address and Data Buses | . 1-9 |
| | | 1.4.2 | Special Bus Operations | 1-10 |
| | | 1.4.3 | Alignment of 32-Bit Accesses to Even Addresses | 1-11 |
| 2 | Cent | ral Proc | essing Unit | . 2-1 |
| | Desc | ribes the | registers and primary functions of the TMS320C28x CPU. | |
| | 2.1 | CPU A | rchitecture | . 2-2 |
| 2.2 CPU Registers | | legisters | . 2-4 | |
| | | 2.2.1 | Accumulator (ACC, AH, AL) | . 2-6 |
| | | 2.2.2 | Multiplicand Register (XT) | . 2-8 |
| | | 2.2.3 | Product Register (P, PH, PL) | . 2-9 |
| | | 2.2.4 | Data Page Pointer (DP) | 2-10 |
| | | 2.2.5 | Stack Pointer (SP) | 2-11 |
| | | 2.2.6 | Auxiliary Registers (XAR0-XAR7, AR0-AR7) | 2-12 |
| | | 2.2.7 | Program Counter (PC) | 2-14 |
| | | 2.2.8 | Return Program Counter (RPC) | 2-14 |
| | | 2.2.9 | Status Registers (ST0, ST1) | 2-14 |
| | | 2.2.10 | Interrupt-Control Registers (IFR, IER, DBGIER) | 2-14 |
| | 2.3 | Status | Register (ST0) | 2-16 |
| | 2.4 | Status | Register ST1 | 2-34 |
| | 2.5 | Progra | m Flow | 2-39 |

| | 2.6 2.7 | 2.5.1 2.5.2 2.5.3 2.5.4 Multiply 2.6.1 2.6.2 Shift Op | Interrupts Branches, Calls, and Returns Repeating a Single Instruction Instruction Pipeline Operations 16-bit X 16-bit Multiplication 32-Bit X 32-Bit Multiplication | 2-39 2-39 2-39 2-40 2-41 2-41 2-42 2-44 |
|---|-----------------|--|--|--|
| 3 | CPU I | nterrup | ts and Reset | 3-1 |
| | Descr explai | ibes the ins the e | TMS320C28x interrupts and how they are handled by the CPU. Also ffects of a hardware reset. | |
| | 3.1 | CPU In | terrupts Overview | 3-2 |
| | 3.2 | CPU In | terrupt Vectors and Priorities | 3-4 |
| | 3.3 | Maskat | ble Interrupts: INT1-INT14, DLOGINT, and RTOSINT | 3-6 |
| | | 3.3.1 | CPU Interrupt Flag Register (IFR) | 3-7 |
| | | 3.3.2 | CPU Interrupt Enable Register (IER) and CPU Debug Interrupt Enable Register (DBGIER) | 3-8 |
| | 3.4 | Standa | rd Operation for Maskable Interrupts | 3-11 |
| | 3.5 | Nonma | skable Interrupts | 3-17 |
| | | 3.5.1 | INTR Instruction | 3-17 |
| | | 3.5.2 | TRAP Instruction | 3-18 |
| | | 3.5.3 | Hardware Interrupt NMI | 3-21 |
| | 3.6 | Illegal-I | | 3-22 |
| | 3.7 | Harowa | | 3-23 |
| 4 | Pipeli | ne | | 4-1 |
| | Descr | ibes the | phases and operation of the instruction pipeline. | |
| | 4.1 | Pipelini | ng of Instructions | 4-2 |
| | | 4.1.1 | Decoupled Pipeline Segments | 4-4 |
| | | 4.1.2 | Instruction-Fetch Mechanism | 4-4 |
| | | 4.1.3 | Address Counters FC, IC, and PC | 4-5 |
| | 4.2 | VISUAIIZ | zing Pipeline Activity | 4-7 |
| | 4.3 | | Mait States | 4-10 |
| | | 432 | Instruction-Not-Available Condition | 4-10 |
| | 4.4 | Pipeline | Protection | 4-12 |
| | | 4.4.1 | Protection During Reads and Writes to the Same Data-Space Location | 4-12 |
| | | 4.4.2 | Protection Against Register Conflicts | 4-13 |
| | 4.5 | Avoidin | g Unprotected Operations | 4-16 |
| | | 4.5.1 | Unprotected Program-Space Reads and Writes | 4-16 |
| | | 4.5.2 | An Access to One Location That Affects Another Location | 4-16 |
| | | 4.5.3 | vvrite Followed By Read Protection Mode | 4-17 |
| 5 | C28x | Addres | sing Modes | 5-1 |
| | Descr | ibes the | addressing modes of the C28x. | |
| | 5.1 | Types of | of Addressing Modes | 5-2 |

| | 5.2 | Addressing Modes Select Bit (AMODE) | |
|---|-------|--|---|
| | 5.3 | Assembler/Compiler Tracking of AMODI | E Bit 5-7 |
| | 5.4 | Direct Addressing Modes (DP) | |
| | 5.5 | Stack Addressing Modes (SP) | 5-9 |
| | 5.6 | Indirect Addressing Modes | 5-10 |
| | | 5.6.1 C28x Indirect Addressing Modes | s (XAR0 to XAR7) 5-10 |
| | | 5.6.2 C2xLP Indirect Addressing Mod | es (ARP, XAR0 to XAR7) 5-12 |
| | | 5.6.3 Circular Indirect Addressing Mod | des (XAR6, XAR1) 5-21 |
| | 5.7 | Register Addressing Modes | |
| | | 5.7.1 32-Bit Register Addressing Mod | es |
| | | 5.7.2 16-Bit Register Addressing Mod | es |
| | 5.8 | Data/Program/IO Space Immediate Add | ressing Modes 5-28 |
| | 5.9 | Program Space Indirect Addressing Mod | des 5-30 |
| | 5.10 | Byte Addressing Modes | |
| | 5.11 | Alignment of 32-Bit Operations | |
| e | C00v | Accomply Longuage Instructions | 6.1 |
| 0 | 028X | Assembly Language Instructions | o-1 |
| | Prese | ents summaries of the instruction set, dent | nes special symbols and notations used, and |
| | uesch | | |
| | 6.1 | Instruction Set Summary (Organized by | Function) |
| | 6.2 | Register Operations | |
| 7 | Emula | ation Features | |
| | Expla | ins features supported by the T320C2800 |) CPU for testing and debugging programs. |
| | 71 | Overview of Emulation Features | 7.2 |
| | 7.1 | Dobug Interface | |
| | 73 | | |
| | 7.3 | Execution Control Modes | |
| | 7.4 | Z 4 1 Stop Mode | ······································ |
| | | 7.4.1 Otop Mode | 70 |
| | | 7.4.2 Real-Time Mode | al Time Mode 7 11 |
| | 75 | Aborting Interrupts With the ABORTI Ins | truction 7-15 |
| | 7.5 | DT-DMA Mechanism | 7-16 |
| | 7.0 | Analysis Breaknoints Watchnoints and | Counter(s) 7-10 |
| | 1.1 | 7 7 1 Analysis Breakpoints, Watchpoints, and | 7-10 |
| | | 7.7.2 Watchpoints | 7 10 |
| | | 7.7.3 Benchmark Counter/Event Court | nter(e) 7-20 |
| | | 7.7.4 Typical Analysis Unit Configurat | ione 7-20 |
| | 78 | Data Logging | 7-23 |
| | 7.0 | 7.8.1 Creating a Data Logging Transfe | ar Buffar 7.23 |
| | | 7.8.2 Accessing the Emulation Regist | ars Pronarly 7.20 |
| | | | /-/0 |
| | | | דמ ד דמ ד |
| | | 7.8.3 Data Log Interrupt (DLOGINT) | |
| | 7.0 | 7.8.3 Data Log Interrupt (DLOGINT) 7.8.4 Examples of Data Logging Sharing Analysis Resources | |

| Contents |
|----------|
|----------|

| | 7.10 | Diagnostics and Recovery | 7-31 |
|---|---|---|---|
| Α | Regis Is a c TMS3 | ster Quick Reference concise, central resource for information about the status and control registers of the 320C28x CPU. | A-1 |
| | A.1 A.2 | Reset Values of and Instructions for Accessing the Registers | A-2 A-3 |
| В | Subr Expla chip F B.1 B.2 | nitting ROM Codes to TI ins the process for submitting custom program code to TI for designing masks for the on-ROM on a TMS320 DSP. Introduction Code Submission | B-1 B-2 B-4 |
| | B.3 B.4 | ROM Layout ROM Code Generation Flow | B-5 B-6 |
| С | C2xL C.1 | P and C28x Architectural Differences Summary of Architecture Differences Between C2xLP and C28x C.1.1 Enhancements of the C28x over the C2xLP: | C-1 C-2 C-2 |
| | C.2 | Registers C.2.1 CPU Register Changes C.2.2 Data Page (DP) Pointer Changes C.2.3 Status Register Changes C.2.4 Register Reset Conditions | C-3 C-4 C-5 C-7 C-10 |
| | C.3 | Memory Map C | C-12 |
| D | C2xL D.1 D.2 D.3 D.4 | P Migration Guidelines Introduction Recommended Migration Flow Mixing C2xLP and C28x Assembly Code Examples D.4.1 Boot Code for C28x operating mode initalization D.4.2 IER/IFR Code D.4.3 Context Save/Restore | D-1 D-2 D-3 D-6 D-7 D-7 D-7 D-7 |
| | D.5 | Reference Tables for C2xLP Code Migration Topics | D-10 |
| Е | C2xL Desci | P Instruction Set Compatibility ribes the instruction set compatibility between the C2xLP and the C28x. | E-1 |
| | E.1 E.2 E.3 | Condition Tests on Flags C2xLP vs. C28x Mnemonics Repeatable Instructions | E-2 E-3 E-9 |
| F | Migra F.1 | ation From C27x to C28x Architecture Changes F.1.1 Changes to Registers | F-1 F-2 F-2 |

| | F12 | Full Context Save and Bestore | F-5 |
|-----|---------|----------------------------------|-------|
| | T.T.Z | | T = 0 |
| | F.1.3 | B0/B1 Memory Map Consideration | ⊦-6 |
| | F.1.4 | C27x Object Compatibility | F-8 |
| F.2 | Moving | g to a C28x Object | F-9 |
| | F.2.1 | Caution When Changing OJBMODE | F-9 |
| F.3 | Migrati | ng to C28x Object Code | F-11 |
| | F.3.1 | Instruction Syntax Changes | F-11 |
| | F.3.2 | Repeatable Instructions | F-13 |
| | F.3.3 | Changes to the SUBCU Instruction | F-14 |
| F.4 | Compi | ling C28x Source Code | F-16 |

Figures

| | Lligh Lovel Concentual Diagram of the CDU |
|-------------|---|
| 1-1. | TMS220C29v High Level Memory Men |
| 1-2. 0 1 | Conceptual Block Diagram of the CPU |
| 2-1. | Conceptual block blagram of the CFO |
| 2-2. | DZOX Registers |
| 2-3. | Individually Accessible Ponions of the Accumulator |
| 2-4. | Individually Accessible Halves of the D Degister |
| 2-5. | Degee of Deta Memory |
| 2-0. | Pages of Data Memory |
| 2-7. | Address Reach of the Stack Pointer |
| 2-8. | XAR0 – XAR7 Registers |
| 2-9. | XARU – XAR7 2-13 Dia Eistala of Obstata Devisition (OTO) 0.40 |
| 2-10. | Bit Fields of Status Register (STU) |
| 2-11. | Bit Fields of Status Register 1 (ST1) |
| 2-12. | Conceptual Diagram of Components Involved in 16 X 10-Bit Multiplication |
| 2-13. | Conceptual Diagram of Components involved in 32 X 32-Bit Multiplication |
| 3-1. | Interrupt Flag Register (IFR) |
| 3-2. | Interrupt Enable Register (IER) |
| 3-3. | Debug Interrupt Enable Register (DBGIER) |
| 3-4. | Standard Operation for CPU Maskaple Interrupts |
| 3-5. | Functional Flow Chart for an Interrupt Initiated by the TRAP Instruction |
| 5-1. | Circular Buffer with AMODE = 0 |
| 5-2. | Circular Buffer with AMODE = 1 |
| 7-1. | JIAG Header to Interface a Target to the Scan Controller |
| 7-2. | Stop Mode Execution States |
| 7-3. | Real-time Mode Execution States |
| 7-4. | Stop Mode Versus Real-Time Mode |
| 7-5. | Process for Handling a DT-DMA Request |
| 7-6. | ADDRL (at Data-Space Address 00 083816) |
| 7-7. | ADDRH (at Data-Space Address 00 083916) |
| 7-8. | REFL (at Data-Space Address 00 084A16) |
| 7-9. | REFH (at Data-Space Address 00 084B16) |
| 7-10. | Valid Combinations of Analysis Resources |
| A-1. | Status register S10 A-4 |
| A-2. | Status register S11, Bits15–8 A-5 |
| A–3. | Status Register S11, Bits 7–0 A-6 |
| A-4. | Interrupt flag register (IFR) A-7 |

| A–5. | Interrupt enable register (IER) | A-8 |
|------|---|------|
| A-6. | Debug interrupt enable register (DBGIER) | A-9 |
| B–1. | TMS320 ROM Code Prototype and Production Flowchart | B-3 |
| C–1. | Register Changes From C2xLP to C28x | C-3 |
| C–2. | Direct Addressing Mode Mapping | C-6 |
| C–3. | Status Register Comparison Between C2xLP and C28x | C-7 |
| C-4. | Memory Map Comparison (See Note A) | C-13 |
| D-1. | Flow Chart of Recommended Migration Steps | D-4 |
| F–1. | C28x Registers | F-2 |
| F–2. | Full Context Save/Restore | F-5 |
| F–3. | Code for a Full Context Save/Restore for C28x vs C27x | F-6 |
| F-4. | Mapping of Memory Blocks B0 and B1 on C27x | F-7 |
| F–5. | C27x Compatible Mapping of Blocks M0 and M1 | F-7 |
| F–6. | Building a C27x Object File From C27x Source | F-8 |
| F–7. | Building a C28x Object File From Mixed C27x/C28x Source | F-9 |
| F-8. | Compiling C28x Source | F-16 |

Tables

| 1_1 | Compatibility Modes | 1-2 |
|-------|---|-------|
| 1_2 | Summary of Bus Use During Data-Space and Program-Space Accesses | 1_10 |
| 1_3 | Special Bus Operations | 1-11 |
| 2_1 | CPU Register Summary | 2-4 |
| 2_2 | Available Operations for Shifting Values in the Accumulator | 2-8 |
| 2_3 | Product Shift Modes | 2-10 |
| 2_4 | Instructions That Affect OV/C/OVCU | 2-17 |
| 2-5 | Instructions Affected by the PM Bits | 2-20 |
| 2-6 | Instructions Affected by V flag | 2-21 |
| 2-7 | Negative Flag Under Overflow Conditions | 2-24 |
| 2-8 | Bits Affected by the C Bit | 2-25 |
| 2-9 | Instructions That Affect the TC Bit | 2-31 |
| 2–10. | Instructions Affected by SXM | 2-33 |
| 2–11. | Shift Operations | 2-45 |
| 3–1. | Interrupt Vectors and Priorities | . 3-4 |
| 3–2. | Requirements for Enabling a Maskable Interrupt | . 3-7 |
| 3–3. | Register Pairs Saved and SP Positions for Context Saves | 3-14 |
| 3–4. | Register Pairs Saved and SP Positions for Context Saves | 3-20 |
| 3–5. | Registers After Reset | 3-23 |
| 5–1. | Addressing Modes for "loc16" or "loc32" | . 5-4 |
| 6–1. | Instruction Set Summary (Organized by Function) | . 6-2 |
| 6–2. | Register Operations | . 6-4 |
| 7–1. | 14-Pin Header Signal Descriptions | . 7-4 |
| 7–2. | Selecting Device Operating Modes By Using TRST, EMU0, and EMU1 | . 7-5 |
| 7–3. | Interrupt Handling Information By Mode and State | 7-13 |
| 7–4. | Start Address and DMA Registers | 7-25 |
| 7–5. | End-Address Registers | 7-26 |
| 7–6. | Analysis Resources | 7-30 |
| A–1. | Reset Values of the Status and Control Registers | . A-2 |
| B–1. | Checksum Computation Memory Locations | . B-7 |
| C-1. | General Features | . C-2 |
| C-2. | C2xLP Product Mode Shifter | . C-8 |
| C–3. | C28x Product Mode Shifter | . C-8 |
| C-4. | Reset Conditions of Internal Registers | C-10 |
| C-5. | Status Register Bits | C-11 |
| C-6. | B0 Memory Map | C-14 |

| D–1. | Code to Save Contents Of IMR (IER) And Disabling Lower Priority Interrupts At | |
|-------|---|------|
| | Beginning Of ISR | D-7 |
| D–2. | Code to Disable an Interrupt | D-7 |
| D–3. | Code to Enable an Interrupt | D-8 |
| D-4. | Code to Clear the IFR Register | D-8 |
| D–5. | Full Context Save/Restore Comparison | D-9 |
| D-6. | C2xLP and C28x Differences in Interrupts | D-10 |
| D-7. | C2xLP and C28x Differences in Status Registers | D-11 |
| D-8. | C2xLp and C28x Differences in Memory Maps | D-12 |
| D-9. | C2xLP and C28x Differences in Instructions and Registers | D-13 |
| D–10. | Code Generation Tools and Syntax Differences | D-15 |
| E–1. | C28x and C2xLP Flags | E-2 |
| E-2. | C2xLP Instructions and C28x Equivalent Instructions | E-3 |
| E–3. | Repeatable Instructions for the C2xLP and C28x | E-9 |
| F–1. | STO Register Bits | F-3 |
| F–2. | ST1 Register Bits | F-4 |
| F–3. | Instruction Syntax Change | F-12 |

Examples

| 3–1. | Typical ISR |
|------|---|
| 4–1. | Relationship Between Pipeline and Address Counters FC, IC, and PC 4-6 |
| 4–2. | Diagramming Pipeline Activity 4-8 |
| 4–3. | Simplified Diagram of Pipeline Activity 4-9 |
| 4-4. | Conflict Between a Read From and a Write to Same Memory Location 4-13 |
| 4–5. | Register Conflict 4-14 |
| 7–1. | Initialization Code for Data Logging With Word Counter |
| 7–2. | Initialization Code for Data Logging With End Address |
| | |

Chapter 1

Architectural Overview

The TMS320C28x[™] is one of several fixed-point generations of digital signal processors (DSPs) in the TMS320 family. The C28x[™] is source-code and object-code compatible with the C27x[™]. In addition, much of the code written for the C2xLP CPU can be reassembled to run on a C28x device.

The C2xLP CPU is used in all TMS320F24xx and TMS320C20x DSPs and their derivatives. This document refers to C2xLP as a generic name for the DSP CPU used in these devices.

This chapter provides an overview of the architectural structure and components of the C28x CPU.

Topic

Page

| 1.1 | Introduction to the CPU 1-2 |
|-----|-----------------------------|
| 1.2 | Components of the CPU 1-4 |
| 1.3 | Memory Map 1-7 |
| 1.4 | Memory Interface1-9 |

1.1 Introduction to the CPU

The CPU is a low-cost 32-bit fixed-point digital signal processor (DSP). This device draws from the best features of digital signal processing; reduced instruction set computing (RISC); and microcontroller architectures, firmware, and tool sets. The DSP features include a modified Harvard architecture and circular addressing. The RISC features are single-cycle instruction execution, register-to-register operations, and modified Harvard architecture (usable in Von Neumann mode). The microcontroller features include ease of use through an intuitive instruction set, byte packing and unpacking, and bit manipulation.

The modified Harvard architecture of the CPU enables instruction and data fetches to be performed in parallel. The CPU can read instructions and data while it writes data simultaneously to maintain the single-cycle instruction operation across the pipeline. The CPU does this over six separate address/data buses.

1.1.1 Compatibility With Other TMS320 CPUs

The C28x DSP features compatibility modes that minimize the migration effort from the C27x and C2xLP CPUs. The operating mode of the device is determined by a combination of the OBJMODE and AMODE bits in status register 1 (ST1) as shown in Table 1–1. The OBJMODE bit allows you to select between code compiled for a C28x (OBJMODE == 1) and code compiled for a C27x (OBJMODE == 0). The AMODE bit allows you to select between C28x/C27x instruction addressing modes (AMODE == 0) and C2xLP compatible instruction addressing modes (AMODE == 1).

Table 1–1. Compatibility Modes

| | OBJMODE | AMODE |
|--|---------|-------|
| C28x Mode | 1 | 0 |
| C2xLP Source-compatible Mode | 1 | 1 |
| C27x Object-compatible Mode ⁺ | 0 | 0 |

[†] The C28x is in C27x-compatible mode at reset.

- C28x Mode: In C28x mode, you can take advantage of all the C28x native features, addressing modes, and instructions. To operate in C28x mode from reset, your code must first set the OBJMODE bit by using the "C28OBJ" (or "SETC OBJMODE") instruction. This book assumes you are operating in C28x mode unless stated otherwise.
- C2xLP Source-Compatible Mode: C2xLP source-compatible mode allows you to run C2xLP source code which has been reassembled using

the C28x code-generation tools. For more information on operating in this mode and migration from a C2xLP CPU, see Appendices C, D, and E.

C27x Object-Compatible Mode: At reset, the C28x CPU operates in C27x object-compatible mode. In this mode, the C28x is 100% object-code and cycle-count compatible with the C27x CPU. For detailed information on operating in C27x object-compatible mode and migrating from the C27x, see Appendix F.

1.1.2 Switching to C28x Mode From Reset

At reset, the C28x CPU is in C27x Object-Compatible Mode (OBJMODE == 0, AMODE == 0) and is 100% compatible with the C27x CPU. To take advantage of the enhanced C28x instruction set, you must instead operate the device in C28x mode. To do this, after a reset your code must first set the OBJ-MODE bit in ST1 by using the "C28OBJ" (or "SETC OBJMODE") instruction.

1.2 Components of the CPU

As shown in Figure 1–1, the CPU contains:

- A CPU for generating data- and program-memory addresses; decoding and executing instructions; performing arithmetic, logical, and shift operations; and controlling data transfers among CPU registers, data memory, and program memory
- Emulation logic for monitoring and controlling various parts and functionalities of the DSP and for testing device operation
- Signals for interfacing with memory and peripherals, clocking and controlling the CPU and the emulation logic, showing the status of the CPU and the emulation logic, and using interrupts

The CPU does not contain memory, a clock generator, or peripheral devices. For information about interfacing to these items, see the *C28x Peripheral User's Guide* (literature number SPRU566) and the data sheet that corresponds to your DSP.

Figure 1–1. High-Level Conceptual Diagram of the CPU



1.2.1 Central Processing Unit (CPU)

The CPU is discussed in more detail in Chapter 2, but following is a list of its major features:

- Protected pipeline. The CPU implements an 8-phase pipeline that prevents a write to and a read from the same location from occurring out of order.
- Independent register space. The CPU contains registers that are not mapped to data space. These registers function as system-control

registers, math registers, and data pointers. The system-control registers are accessed by special instructions. The other registers are accessed by special instructions or by a special addressing mode (register addressing mode).

- Arithmetic logic unit (ALU). The 32-bit ALU performs 2s-complement arithmetic and Boolean logic operations.
- Address register arithmetic unit (ARAU). The ARAU generates datamemory addresses and increments or decrements pointers in parallel with ALU operations.
- Barrel shifter. This shifter performs all left and right shifts of data. It can shift data to the left by up to 16 bits and to the right by up to 16 bits.
- Multiplier. The multiplier performs 32-bit × 32-bit 2s-complement multiplication with a 64-bit result. The multiplication can be performed with two signed numbers, two unsigned numbers, or one signed number and one unsigned number.

1.2.2 Emulation Logic

The emulation logic includes the following features. For more details about these features, see Chapter 7, *Emulation Features*.

- Debug-and-test direct memory access (DT-DMA). A debug host can gain direct access to the content of registers and memory by taking control of the memory interface during unused cycles of the instruction pipeline.
- Data logging. The emulation logic enables application-initiated transfers of memory contents between the C28x and a debug host.
- A counter for performance benchmarking
- Multiple debug events. Any of the following *debug events* can cause a break in program execution:
 - A breakpoint initiated by the ESTOP0 or ESTOP1 instruction
 - An access to a specified program-space or data-space location
 - A request from the debug host or other hardware

When a debug event causes the C28x to enter the debug-halt state, the event is called a *break event*.

Real-time mode of operation. When the C28x is in this mode and a break event occurs, the main body of program code comes to a halt, but time-critical interrupts can still be serviced.

1.2.3 Signals

The CPU has four main types of signals:

- Memory-interface signals. These signals transfer data among the CPU, memory, and peripherals; indicate program-memory accesses and datamemory accesses; and differentiate between accesses of different sizes (16-bit or 32-bit).
- Clock and control signals. These provide clocking for the CPU and the emulation logic, and they are used to control and monitor the CPU.
- Reset and interrupt signals. These are used for generating a hardware reset and interrupts, and for monitoring the status of interrupts.
- Emulation signals. These signals are used for testing and debugging.

1.3 Memory Map

The CPU contains no memory, but can access memory elsewhere on the C28x DSP or outside the DSP.

The C28x uses 32-bit data addresses and 22-bit program addresses. This allows for a total address reach of 4G words (1 word = 16 bits) in data space and 4M words in program space. Memory blocks on all C28x designs are uniformly mapped to both program and data space. Figure 1–2 shows a high-level view of how addresses are allocated in program space and data space.

The memory map in Figure 1–2 has been divided into the following segments:

- On-chip program/data
- Reserved
- CPU interrupt vectors

For specific details about each of the map segments, see the data sheet for your DSP. See Appendix D for more information on the C2xLP compatible memory space.

1.3.1 On-Chip Program/Data

All C28x-based CPU devices contain two blocks of single access on-chip memory referred to as M0 and M1. Each of these blocks is 1K words in size. M0 is mapped at addresses 00 $0000_{16} - 00 03FF_{16}$ and M1 is mapped at addresses 00 $0400_{16} - 00 07FF_{16}$. Like all other memory blocks on the C28x devices, M0 and M1 are mapped to both program and data space. Therefore, you can use M0 and M1 to execute code or for data variables. At reset, the stack pointer is set to the top of block M1.

Depending on the device, it may also have additional random-access memory (RAM), read-only memory (ROM), or flash memory.

1.3.2 Reserved

Addresses 0000 0800–0000 09FF in data space are reserved for CPU Emulation Registers on all C28x designs.

1.3.3 CPU Interrupt Vectors

Sixty-four addresses in program space are set aside for a table of 32 CPU interrupt vectors. The CPU vectors can be mapped to the top or bottom of program space by way of the VMAP bit. For more information about the CPU vectors, see Section 3.2, *Interrupt Vectors and Priorities* on page 3-4.

For devices with a peripheral interrupt expansion (PIE) block, the interrupt vectors will reside in the PIE vector table and this memory can be used as program memory.



Figure 1-2. TMS320C28x High-Level Memory Map

See the data sheet for your specific device for details of the exact memory map.

1.4 Memory Interface

The C28x memory map is accessible outside the CPU by the memory interface, which connects the CPU logic to memories, peripherals, or other interfaces. The memory interface includes separate buses for program space and data space. This means an instruction can be fetched from program memory while data memory is being accessed.

The interface also includes signals that indicate the type of read or write being requested by the CPU. These signals can select a specified memory block or peripheral for a given bus transaction. In addition to 16-bit and 32-bit accesses, the C28x supports special byte-access instructions which can access the least significant byte (LSByte) or most significant byte (MSByte) of an addressed word. Strobe signals indicate when such an access is occurring on a data bus.

1.4.1 Address and Data Buses

The memory interface has three address buses:

- **PAB** *Program address bus.* The PAB carries addresses for reads and writes from program space. PAB is a 22-bit bus.
- **DRAB** *Data-read address bus.* The 32-bit DRAB carries addresses for reads from data space.
- **DWAB** *Data-write address bus.* The 32-bit DWAB carries addresses for writes to data space.

The memory interface also has three data buses:

- **PRDB** *Program-read data bus.* The PRDB carries instructions or data during reads from program space. PRDB is a 32-bit bus.
- **DRDB** *Data-read data bus.* The DRDB carries data during reads from data space. PRDB is a 32-bit bus.
- **DWDB** *Data-/Program-write data bus.* The 32-bit DWDB carries data during writes to data space or program space.

Table 1-2 summarizes how these buses are used during accesses.

Table 1–2. Summary of Bus Use During Data-Space and Program-Space Accesses

| Access Type | Address Bus | Data Bus |
|-------------------------|-------------|----------|
| Read from program space | PAB | PRDB |
| Read from data space | DRAB | DRDB |
| Write to program space | PAB | DWDB |
| Write to data space | DWAB | DWDB |

A program-space read and a program-space write cannot happen simultaneously because both use the PAB. Similarly, a program-space write and a data-space write cannot happen simultaneously because both use the DWDB. Transactions that use different buses can happen simultaneously. For example, the CPU can read from program space (using PAB and PRDB), read from data space (using DRAB and DRDB), and write to data space (using DWAB and DWDB) at the same time.

1.4.2 Special Bus Operations

Typically, PAB and PRDB are used only for reading instructions from program space, and DWDB is used only for writing data to data space. However, the instructions in Table 1–3 are exceptions to this behavior. For more details about using these instructions, see Chapter 6, *Assembly Language Instructions*.

| Table 1–3. Special Bus | С | perations |
|------------------------|---|-----------|
|------------------------|---|-----------|

| - | |
|---------------------------------|--|
| Instruction | Special Bus Operation |
| PREAD | This instruction reads a data value rather than an instruction from pro- gram space. It then transfers that value to data space or a register. |
| | For the read from program space, the CPU places the source address on the program address bus (PAB), sets the appropriate program- space select signals, and reads the data value from the program-read data bus (PRDB). |
| PWRITE | This instruction writes a data value to program space. The value is read from from data space or a register. |
| | For the write to program space, the CPU places the destination ad- dress on the program address bus (PAB), sets the appropriate pro- gram-space select signals, and writes the data value to the data-/pro- gram-write data bus (DWDB). |
| MAC DMAC | As part of their operation, these instructions multiply two data values, one of which is read from program space. |
| QMACL IMACL XMAC XMACD | For the read from program space, the CPU places the program-space source address on the program address bus (PAB), sets the appropri- ate program-space select signals, and reads the program data value from the program read data bus (PRDB). |

1.4.3 Alignment of 32-Bit Accesses to Even Addresses

The C28x CPU expects memory wrappers or peripheral-interface logic to align any 32-bit read or write to an even address. If the address-generation logic generates an odd address, the CPU must begin reading or writing at the previous even address. This alignment does not affect the address values generated by the address-generation logic.

Most instruction fetches from program space are performed as 32-bit read operations and are aligned accordingly. However, alignment of instruction fetches are effectively invisible to a programmer. When instructions are stored to program space, they do not have to be aligned to even addresses. Instruction boundaries are decoded within the CPU.

You need to be concerned with alignment when using instructions that perform 32-bit reads from or writes to data space.

Chapter 2

Central Processing Unit

The central processing unit (CPU) is responsible for controlling the flow of a program and the processing of instructions. It performs arithmetic, Boolean-logic, multiply, and shift operations. When performing signed math, the CPU uses 2s-complement notation. This chapter describes the architecture, registers, and primary functions of the CPU.

Topic Page 2.1 CPU Registers 2-4 2.2 Status Register ST0 2-16 2.3 Status Register ST1 2-34 2.4 2.5 Multiply Operations 2-41 2.6 Shift Operations 2-44 2.7

2.1 CPU Architecture

All C28x devices contain a central processing unit (CPU), emulation logic, and signals for interfacing with memory and peripherals. Included with these signals are three address buses and three data buses. Figure 2–1 shows the major blocks and data paths of the C28x CPU. It does not reflect the actual silicon implementation. The shaded buses are memory-interface buses that are external to the CPU. The operand bus supplies the values for multiplier, shifter, and ALU operations, and the result bus carries the results to registers and memory. The main blocks of the CPU are:

- Program and data control logic. This logic stores a queue of instructions that have been fetched from program memory.
- Real-Time emulation and visibility
- Address register arithmetic unit (ARAU). The ARAU generates addresses for values that must be fetched from data memory. For a data read, it places the address on the data-read address bus (DRAB); for a data write, it loads the data-write address bus (DWAB). The ARAU also increments or decrements the stack pointer (SP) and the auxiliary registers (XAR0, XAR1, XAR2, XAR3, XAR4, XAR5, XAR6, and XAR7).
- Atomic arithmetic logic unit (ALU). The 32-bit ALU performs 2s-complement arithmetic and Boolean logic operations. Before doing its calculations, the ALU accepts data from registers, from data memory, or from the program control logic. The ALU saves results to a register or to data memory.
- Prefetch queue and instruction decode
- Address generators for program and data
- Fixed-point MPY/ALU. The multiplier performs 32-bit × 32-bit 2s-complement multiplication with a 64-bit result. In conjunction with the multiplier, the '28xx uses the 32-bit multiplicand register (XT), the 32-bit product register (P), and the 32-bit accumulator (ACC). The XT register supplies one of the values to be multiplied. The result of the multiplication can be sent to the P register or to ACC.
- Interrupt processing



Figure 2-1. Conceptual Block Diagram of the CPU

Central Processing Unit 2-3

2.2 CPU Registers

Table 2–1 lists the main CPU registers and their values after reset. Sections 2.2.1 through 2.2.10 describe the registers in more detail. Figure 2–2 shows the registers.

Table 2–1. CPU Register Summary

| Register | Size | Description | Value After Reset |
|----------|---------|----------------------|-------------------|
| ACC | 32 bits | Accumulator | 0x0000000 |
| AH | 16 bits | High half of ACC | 0x0000 |
| AL | 16 bits | Low half of ACC | 0x0000 |
| XAR0 | 16 bits | Auxiliary register 0 | 0x0000000 |
| XAR1 | 32 bits | Auxiliary register 1 | 0x0000000 |
| XAR2 | 32 bits | Auxiliary register 2 | 0x0000000 |
| XAR3 | 32 bits | Auxiliary register 3 | 0x0000000 |
| XAR4 | 32 bits | Auxiliary register 4 | 0x0000000 |
| XAR5 | 32 bits | Auxiliary register 5 | 0x0000000 |
| XAR6 | 32 bits | Auxiliary register 6 | 0x0000000 |
| XAR7 | 32 bits | Auxiliary register 7 | 0x0000000 |
| AR0 | 16 bits | Low half of XAR0 | 0x0000 |
| AR1 | 16 bits | Low half of XAR1 | 0x0000 |
| AR2 | 16 bits | Low half of XAR2 | 0x0000 |
| AR3 | 16 bits | Low half of XAR3 | 0x0000 |
| AR4 | 16 bits | Low half of XAR4 | 0x0000 |
| AR5 | 16 bits | Low half of XAR5 | 0x0000 |
| AR6 | 16 bits | Low half of XAR6 | 0x0000 |
| AR7 | 16 bits | Low half of XAR7 | 0x0000 |

| Register | Size | Description | Value After Reset |
|----------|---------|------------------------------------|--|
| DP | 16 bits | Data-page pointer | 0x0000 |
| IFR | 16 bits | Interrupt flag register | 0x0000 |
| IER | 16 bits | Interrupt enable register | 0x0000 (INT1 to INT14, DLOGINT, RTOSINT disabled) |
| DBGIER | 16 bits | Debug interrupt enable register | 0x0000 (INT1 to INT14, DLOGINT, RTOSINT disabled) |
| Р | 32 bits | Product register | 0x0000000 |
| PH | 16 bits | High half of P | 0x0000 |
| PL | 16 bits | Low half of P | 0x0000 |
| PC | 22 bits | Program counter | 0x3F FFC0 |
| RPC | 22 bits | Return program counter | 0x0000000 |
| SP | 16 bits | Stack pointer | 0x0400 |
| ST0 | 16 bits | Status register 0 | 0x0000 |
| ST1 | 16 bits | Status register 1 | 0x080B [†] |
| | | | |
| ХТ | 32 bits | Multiplicand register | 0x0000000 |
| Т | 16 bits | High half of XT | 0x0000 |
| TL | 16 bits | Low half of XT | 0x0000 |

Table 2–1. CPU Register Summary (Continued)

[†] Reset value shown is for devices without the VMAP signal and MOM1MAP signal pinned out. On these devices both of these signals are tied high internal to the device.

Figure 2–2. C28x Registers

| T[16] | TL[16] | XT[32] |
|--------|--------|---------|
| PH[16] | PL[16] | P[32] |
| AH[16] | AL[16] | ACC[32] |

| | | SP[16] | | |
|----------|-------|---------|--------------------|----------|
| DP | | [16] | 6/7-bit offset† | |
| ARO | H[16] | AR0[| 16] | XAR0[32] |
| AR1 | H[16] | AR1[16] | | XAR1[32] |
| AR2I | H[16] | AR2[| 16] | XAR2[32] |
| AR3 | H[16] | AR3[| 16] | XAR3[32] |
| AR4I | H[16] | AR4[| 16] | XAR4[32] |
| AR5I | H[16] | AR5[16] | | XAR5[32] |
| AR6 | H[16] | AR6[16] | | XAR6[32] |
| AR7H[16] | | AR7[16] | | XAR7[32] |
| | | PC[22] | | - |
| | | RPC[22] | | - |
| L | | | |] |
| | | STO | [16] |] |
| | | ST1 | [16] | - |
| | | L | | |
| | | [| | r |

| IER[16] |
|------------|
| DBGIER[16] |
| IFR[16] |

[†] A 6-bit offset is used when operating in C28x mode or C27x object-compatible mode.
 A 7-bit offset is used when operating in C2xLP source-compatible mode. The least significant bit of the DP is ignored when operating in this mode.

2.2.1 Accumulator (ACC, AH, AL)

The accumulator (ACC) is the main working register for the device. It is the destination for all ALU operations except those which operate directly on memory or registers. ACC supports single-cycle move, add, subtract, and
compare operations from 32-bit-wide data memory. It can also accept the 32-bit result of a multiplication operation.

The halves and quarters of the ACC can also be accessed (see Figure 2–3). ACC can be treated as two independent 16-bit registers: AH (high 16 bits) and AL (low 16 bits). The bytes within AH and AL can also be accessed independently. Special byte-move instructions load and store the most significant byte or least significant byte of AH or AL. This enables efficient byte packing and unpacking.





The accumulator has the following associated status bits. For the details on these bits, see section 2.3, *Status Register ST0*.

- Overflow mode bit (OVM)
- Sign-extension mode bit (SXM)
- Test/control flag bit (TC)
- Carry bit (C)
- Zero flag bit (Z)
- Negative flag bit (N)
- Latched overflow flag bit (V)
- Overflow counter bits (OVC)

Table 2–2 shows the ways to shift the content of AH, AL, or ACC.

| Register | Shift Direction | Shift Type | Instruction |
|----------|-----------------|------------|-----------------------------|
| ACC | Left | Logical | LSL or LSLL |
| | | Rotation | ROL |
| | Right | Arithmetic | SFR with SXM = 1 or ASRL |
| | | Logical | SFR with SXM = 0 or LSRL |
| | | Rotation | ROR |
| AH or AL | Left | Logical | LSL |
| | Right | Arithmetic | ASR |
| | | Logical | LSR |

Table 2–2. Available Operations for Shifting Values in the Accumulator

2.2.2 Multiplicand Register (XT)

The multiplicand register (XT register) is used primarily to store a 32-bit signed integer value prior to a 32-bit multiply operation.

The lower 16-bit portion of the XT register is referred to as the TL register. This register can be loaded with a signed 16-bit value that is automatically sign-extended to fill the 32-bit XT register.

The upper 16-bit portion of the XT register is referred to as the T register. The T register is mainly used to store a 16-bit integer value prior to a 16-bit multiply operation.

The T register is also used to specify the shift value for some shift operations. In this case, only a portion of the T register is used, depending on the instruction.

For example:

| ASR AX, T | performs an arithmetic shift right based on the four least significant bits of T: T(3:0) = 015 |
|-------------|--|
| ASRL ACC, T | performs an arithmetic shift right by the five least significant bits of T: T(4:0) 031 |

For these operations, the most significant bits of T are ignored.

Figure 2–4. Individually Accessible Halves of the XT Register



2.2.3 Product Register (P, PH, PL)

The product register (P register) is typically used to hold the 32-bit result of a multiplication. It can also be loaded directly from a 16- or 32-bit data-memory location, a 16-bit constant, the 32-bit ACC, or a 16-bit or a 32-bit addressable CPU register. The P register can be treated as a 32-bit register or as two independent 16-bit registers: PH (high 16 bits) and PL (low 16 bits); see Figure 2–5.

Figure 2–5. Individually Accessible Halves of the P Register



When some instructions access P, PH, or PL, all 32-bits are copied to the ALUshifter block, where the barrel shifter may perform a left shift, a right shift, or no shift. The action of the shifter for these instructions is determined by the product shift mode (PM) bits in status register ST0. Table 2–3 shows the possible PM values and the corresponding product shift modes. When the barrel shifter performs a left shift, the low order bits are filled with zeros. When the shifter performs a right shift, the P register value is sign extended. Instructions that use PH or PL as operands ignore the product shift mode.

For a complete list of instructions affected by PM bits, see Table 2–5 on page 2-20.

| PM Value | Product Shift Mode | | |
|------------------|--|--|--|
| 000 ₂ | Left shift by 1 | | |
| 001 ₂ | No shift | | |
| 010 ₂ | Right shift by 1 | | |
| 011 ₂ | Right shift by 2 | | |
| 100 ₂ | Right shift by 3 | | |
| 101 ₂ | Right shift by 4 (if AMODE = 1, left 4) | | |
| 110 ₂ | Right shift by 5 | | |
| 111 ₂ | Right shift by 6 | | |

| Table 2–3. | Product | Shift | Modes |
|------------|---------|-------|-------|
|------------|---------|-------|-------|

2.2.4 Data Page Pointer (DP)

In the direct addressing modes, data memory is addressed in blocks of 64 words called *data pages*. The lower 4M words of data memory consists of 65 536 data pages labeled 0 through 65 535, as shown in Figure 2–6. In DP direct addressing mode, the 16-bit data page pointer (DP) holds the current data page number. You change the data page by loading the DP with a new number. For information about the direct addressing modes, see section 5.4 on page 5-8.

| Data page | Offset | | Data memory |
|----------------------|---------|-----------|-------------------------|
| 00 0000 0000 0000 00 | 00 0000 | | |
| | ÷ | Page 0: | 0000 0000-0000 003F |
| 00 0000 0000 0000 00 | 11 1111 | | |
| 00 0000 0000 0000 01 | 00 0000 | | |
| • | : | Page 1: | 0000 0040-0000 007F |
| 00 0000 0000 0000 01 | 11 1111 | | |
| 00 0000 0000 0000 10 | 00 0000 | | |
| | | Page 2: | 0000 0080-0000 00BF |
| 00 0000 0000 0000 10 | 11 1111 | _ | |
| | | | |
| | | | |
| | • | | |
| • | | • | |
| | • | • | • |
| • | • | · | • |
| • | · | · | · |
| | 00.0000 | | |
| 11 1111 1111 1111 11 | 00 0000 | | |
| | : | Page 65 8 | 535:003F FFC0-003F FFFF |
| 11 1111 1111 1111 11 | 11 1111 | | |

Figure 2-6. Pages of Data Memory

Data memory above 4M words is not accessible using the DP.

When operating in C2xLP source-compatible mode, a 7-bit offset is used and the least significant bit of the DP register is ignored. See Appendix C for more details.

2.2.5 Stack Pointer (SP)

The stack pointer (SP) enables the use of a software stack in data memory. The stack pointer has only 16 bits and can only address the low 64K of data space (see Figure 2–7). When the SP is used, the upper six bits of the 32-bit address are forced to 0. (For information about addressing modes that use the SP, see section 5.5 on page 5-9.). After reset, SP points to address 0000 0400₁₆.

Figure 2–7. Address Reach of the Stack Pointer



The operation of the stack is as follows:

- The stack grows from low memory to high memory.
- The SP always points to the next empty location in the stack.
- At reset, the SP is initialized, so that it points to address 0000 0400₁₆.
- When 32-bit values are saved to the stack, the least significant 16 bits are saved first, and the most significant 16 bits are saved to the next higher address (little endian format).
- When 32-bit operations read or write a 32-bit value, the C28x CPU expects the memory wrapper or peripheral-interface logic to align that read or write to an even address. For example, if the SP contains the odd address 0000 0083₁₆, a 32-bit read operation reads from addresses 0000 0082₁₆ and 0000 0083₁₆.
- □ The SP overflows if its value is increased beyond FFFF₁₆ or decreased below 0000₁₆. When the SP increases past FFFF₁₆, it counts forward from 0000₁₆. For example, if SP = FFFE₁₆ and an instruction adds 3 to the SP, the result is 0001₁₆. When the SP decreases past 0000₁₆, it counts backward from FFFF₁₆. For example, if SP = 0002₁₆ and an instruction subtracts 4 from SP, the result is FFFE₁₆.
- When values are being saved to the stack, the SP is not forced to align with even or odd addresses. Alignment is forced by the memory wrapper or peripheral-interface logic.

2.2.6 Auxiliary Registers (XAR0–XAR7, AR0–AR7)

The CPU provides eight 32-bit registers that can be used as pointers to memory or as general-purpose registers (see Section 5.6, *Indirect Addressing*

Modes, on page 5-10. The auxiliary registers are: XAR0, XAR1, XAR2, XAR3, XAR4, XAR5, XAR6, and XAR7.

Many instructions allow you to access the 16 LSBs of XAR0–XAR7. As shown in Figure 2–8, the 16 LSBs of the auxiliary registers are referred to as AR0–AR7. AR0–AR7 can be used as general purpose registers for loop control and for efficient 16-bit comparisons.

When accessing AR0–AR7, the upper 16 bits of the register (known as AR0H–AR7H) may or may not be modified, depending on the instruction used (see Chapter 6 for information on the behavior of particular instructions). AR0H–AR7H are accessed only as part of XAR0–XAR7 and are not individually accessible.

Figure 2-8. XAR0 - XAR7 Registers



n = number 0 through 7

For ACC operations, all 32 bits are valid (@XARn). For 16-bit operations, the lower 16 bits are used and upper 16 bits are ignored (@ARn).

XAR0 – XAR7 can also be used by some instructions to point to any value in program memory; see Section 5.6, *Indirect Addressing Modes*.

Many instructions allow you to access the 16 least significant bits (LSBs) of XAR0–XAR7. As shown in Figure 2–9, 16 LSBs of XAR0–XAR7 are known as one auxiliary register of AR0–AR7.





2.2.7 Program Counter (PC)

When the pipeline is full, the 22-bit program counter (PC) always points to the instruction that is currently being processed — the instruction that has just reached the decode 2 phase of the pipeline. Once an instruction reaches this phase of the pipeline, it cannot be flushed from the pipeline by an interrupt. It is executed before the interrupt is taken. The pipeline is discussed in Chapter 4.

2.2.8 Return Program Counter (RPC)

When a call operation is performed using the LCR instruction, the return address is saved in the RPC register and the old value in the RPC is saved on the stack (in two 16-bit operations). When a return operation is performed using the LRETR instruction, the return address is read from the RPC register and the value on the stack is written into the RPC register (in two 16-bit operations). Other call instructions do not use the RPC register. For more information, see the instructions in Chapter 6.

2.2.9 Status Registers (ST0, ST1)

The C28x has two status registers, ST0 and ST1, which contain various flag bits and control bits. These registers can be stored into and loaded from data memory, enabling the status of the machine to be saved and restored for sub-routines.

The status bits have been organized according to when the bit values are modified in the pipeline. Bits in ST0 are modified in the execute phase of the pipeline; bits in ST1 are modified in the decode 2 phase. (For details about the pipeline, see Chapter 4.) The status bits are described in detail in sections 2.3 (ST0) and 2.4 (ST1). Also, ST0 and ST1 are included in Appendix A, *Register Quick Reference*.

2.2.10 Interrupt-Control Registers (IFR, IER, DBGIER)

The C28x CPU has three registers dedicated to the control of interrupts:

- Interrupt flag register (IFR)
- Interrupt enable register (IER)
- Debug interrupt enable register (DBGIER)

These registers handle interrupts at the CPU level. Devices with a peripheral interrupt expansion (PIE) block will have additional interrupt control as part of the PIE module.

The IFR contains flag bits for maskable interrupts (those that can be enabled and disabled with software). When one of these flags is set, by hardware or software, the corresponding interrupt will be serviced if it is enabled. You enable or disable a maskable interrupt with its corresponding bit in the IER. The DBGIER indicates the time-critical interrupts that will be serviced (if enabled) while the DSP is in real-time emulation mode and the CPU is halted.

The C28x CPU interrupts and the interrupt-control registers are described in detail in Chapter 3, *Interrupts*. Also, the IFR, IER, and DBGIER are included in Appendix A, *Register Quick Reference*.

2.3 Status Register (ST0)

The following figure shows the bit fields of status register (ST0). All of these bit fields are modified in the execute phase of the pipeline. Detailed descriptions of these bits follow the figure.

Figure 2–10. Bit Fields of Status Register (ST0)

| 15 | 10 | 9 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|----|-------|---|------|------|------|------|------|------|------|
| OVC/OVCU | | PM | | V | Ν | Z | С | тс | OVM | SXM |
| R/W-00 0000 | | R/W-0 | | RW-0 |

Note: R = Read access; W = Write access; value following dash (-) is value after reset.

OVC/OVCUOverflow counter. The overflow counter behaves differently for signed and unsigned op-
erations.Bits15-10erations.

For signed operations, the overflow counter is a 6-bit signed counter with a range of -32 to 31. When overflow mode is off (OVM = 0), ACC overflows normally, and OVC keeps track of overflows. When overflow mode is on (OVM = 1) and an overflow occurs in ACC, the OVC is not affected. Instead, the CPU automatically fills ACC with a positive or negative saturation value (see the description for OVM on page 2-32).

When ACC overflows in the positive direction (from 7FFF $FFFF_{16}$ to 8000 0000₁₆), the OVC is incremented by 1. When ACC overflows in the negative direction (from 8000 0000₁₆ to 7FFF $FFFF_{16}$) the OVC is decremented by 1. The increment or decrement is performed as the overflow affects the V flag.

For unsigned operations (OVCU), the counter increments for ADD when a Carry is generated and decrements for a SUB when a Borrow is generated (similar to a carry counter).

If OVC increments past its most positive value, 31, the counter wraps around to -32. If OVC decrements past its most negative value, -32, the counter wraps around to 31. At reset, OVC is cleared.

OVC is not affected by overflows in registers other than ACC and is not affected by compare instructions (CMP and CMPL). The table that follows explains how OVC may be affected by the saturate accumulator (SAT ACC) instruction.

Table 2–4 lists the instructions affecting OVC/OVCU. See the instruction set in Chapter 6 for a complete description of each instruction.

| Signed Addition Instructions | Effect on OVC/OVCU |
|---------------------------------|--|
| ADD ACC,loc16 << shift | if(OVM == 0) Inc OVC on +ve signed overflow |
| ADD ACC,#16bit << shift | |
| ADD ACC,loc16 << T | |
| ADD loc16,#16bitSigned | |
| ADDB ACC,#8bit | |
| ADDCL ACC, loc32 | |
| ADDCU ACC,loc16 | |
| ADDL ACC,loc32 | |
| ADDL loc32,ACC | |
| ADDU ACC,loc16 | |
| DMAC ACC:P,loc32,*XAR7/++ | |
| INC loc16 | |
| MAC P,loc16,*XAR7/++ | |
| MAC P,loc16,0:pma | |
| MOVA T,loc16 | |
| MOVAD T,loc16 | |
| MPYA P,loc16,#16bit | |
| MPYA P,T,loc16 | |
| QMACL P,loc32,*XAR7/++ | |
| QMPYAL P,XT,loc32 | |
| SQRA loc16 | |
| XMAC P,loc16,*(pma) | |
| XMACD P,loc16,*(pma) | |
| Signed Subtraction Instructions | Effect on OVC/OVCU |
| DEC loc16 | if(OVM == 0) Dec OVC on -ve signed overflow |
| MOVS T,loc16 | |

Table 2–4. Instructions That Affect OVC/OVCU

| Signed Addition Instructions | Effect on OVC/OVCU | | | |
|------------------------------|--------------------------------|--|--|--|
| MPYS P,T,loc16 | | | | |
| QMPYSL P,XT,loc32 | | | | |
| SBBU ACC,loc16 | | | | |
| SQRS loc16 | | | | |
| SUB ACC,#16bit << shift | | | | |
| SUB ACC,loc16 << shift | | | | |
| SUB ACC,loc16 << T | | | | |
| SUBB ACC,#8bit | | | | |
| SUBBL ACC,loc32 | | | | |
| SUBL ACC,loc32 | | | | |
| SUBL loc32,ACC | | | | |
| SUBRL loc32,ACC | | | | |
| SUBU ACC,loc16 | | | | |
| SUBUL ACC,loc32 | | | | |
| SUBUL P,loc32 | | | | |
| Unsigned Instructions | Effect on OVC/OVCU | | | |
| ADDUL ACC,loc32 | Inc OVC/OVCU on unsigned carry | | | |
| ADDUL P,loc32 | | | | |
| IMPYAL P,XT,loc32 | | | | |
| IMACL P,loc32,*XAR7/++ | | | | |
| Misc Instructions | Effect on OVC/OVCU | | | |
| SAT ACC | if(OVC > 0) Saturate +ve | | | |
| | if(OVC < 0) Saturate -ve | | | |
| | OVC = 0 | | | |
| SAT64 ACC:P | | | | |
| ZAPA | OVC = 0 | | | |
| ZAP OVC | | | | |
| MOV OVC,loc16 | OVC = [loc16(15:10)] | | | |

Table 2-4. Instructions That Affect OVC/OVCU (Continued)

Signed Addition Instructions Effect on OVC/OVCU MOVU OVC,loc16 OVC = [loc16(5:0)]Condition **Operation Performed by SAT ACC Instruction** OVC = 0Leave ACC and OVC unchanged. OVC > 0Saturate ACC in the positive direction (fill ACC with 7FFF FFFF16), and clear OVC. OVC < 0Saturate ACC in the negative direction (fill ACC with 8000 000016), and clear OVC. Product shift mode bits. This 3-bit value determines the shift mode for any output opera-PM tion from the product (P) register. The shift modes are shown in the following table. The out-Bits 9-7 put can be to the ALU or to memory. All instructions that are affected by the product shift mode will sign extend the P register value during a right shift operation. At reset, PM is cleared (left shift by 1 bit is the default). PM is summarized as follows: 000 Left shift by 1. During the shift, the low-order bit is zero filled. At reset, this mode is selected. 001 No shift 010 Right shift by 1. During the shift, the lower bits are lost, and the shifted value is sign extended. Right shift by 2. During the shift, the lower bits are lost, and the shifted value is sign 011 extended. 100 Right shift by 3. During the shift, the lower bits are lost, and the shifted value is sign extended. 101 Right shift by 4. During the shift, the lower bits are lost, and the shifted value is sign extended. Note, if AMODE = 1, then 101 is a left shift by 4. 110 Right shift by 5. During the shift, the lower bits are lost, and the shifted value is sign extended. Right shift by 6. During the shift, the lower bits are lost, and the shifted value is sign 111 extended.

Table 2–4. Instructions That Affect OVC/OVCU (Continued)

Note: For performing unsigned arithmetic, you must use a product shift of 0 (PM = 001) to avoid sign extension and generation of incorrect results.

Table 2–5 lists instructions that are affected by the PM bits. See the instruction set in chapter 6 for a complete description of each instruction.

| Instruction | Effect of PM | | | |
|---------------------------|--------------------------------------|--|--|--|
| CMPL ACC,P << PM | flags set on(ACC – P << PM) | | | |
| DMAC ACC:P,loc32,*XAR7/++ | ACC = ACC + MSW*MSW << PM | | | |
| | P = P + LSW*LSW << PM | | | |
| IMACL P,loc32,*XAR7/++ | P = ([loc32] * Prog[*XAR7/++]) << PM | | | |
| IMPYAL P,XT,loc32 | P = (XT * [loc32]) << PM | | | |
| IMPYL P,XT,loc32 | P = (XT *[loc32]) << PM | | | |
| IMPYSL P,XT,loc32 | ACC = ACC - P unsigned | | | |
| | P = (XT * [loc32]) << PM | | | |
| IMPYXUL P,XT,loc32 | P = (XT sign * [loc32]uns) << PM | | | |
| MAC P,loc16,*XAR7/++ | ACC = ACC + P << PM | | | |
| MAC P,loc16,0:pma | ACC = ACC + P << PM | | | |
| MOV loc16,P | [loc16] = low(P << PM) | | | |
| MOVA T,loc16 | ACC = ACC + P << PM | | | |
| MOVAD T,loc16 | ACC = ACC + P << PM | | | |
| MOVH loc16,P | [loc16] = high(P << PM) | | | |
| MOVP T,loc16 | ACC = P << PM | | | |
| MOVS T,loc16 | ACC = ACC – P << PM | | | |
| MPYA P,loc16,#16bit | ACC = ACC + P << PM | | | |
| MPYA P,T,loc16 | ACC = ACC + P << PM | | | |
| MPYS P,T,loc16 | ACC = ACC – P << PM | | | |
| QMACL P,loc32,*XAR7 | ACC = ACC + P << PM | | | |
| QMACL P,loc32,*XAR7++ | ACC = ACC + P << PM | | | |
| QMPYAL P,XT,loc32 | ACC = ACC + P << PM | | | |
| QMPYSL P,XT,loc32 | ACC = ACC – P << PM | | | |
| SQRA loc16 | ACC = ACC + P << PM | | | |
| SQRS loc16 | ACC = ACC – P << PM | | | |
| XMAC P,loc16,*(pma) | ACC = ACC + P << PM | | | |
| XMACD P,loc16,*(pma) | ACC = ACC + P << PM | | | |

Table 2–5. Instructions Affected by the PM Bits

V Bit 6 **Overflow flag.** If the result of an operation causes an overflow in the register holding the result, V is set and latched. If no overflow occurs, V is not modified. Once V is latched, it remains set until it is cleared by reset or by a conditional branch instruction that tests V. Such a conditional branch clears V regardless of whether the tested condition (V = 0 or V = 1) is true.

An overflow occurs in ACC (and V is set) if the result of an addition or subtraction does not fit within the signed numerical range -2^{31} to $(+2^{31} - 1)$, or 8000 0000₁₆ to 7FFF FFFF₁₆.

An overflow occurs in AH, AL, or another 16-bit register or data-memory location if the result of an addition or subtraction does not fit within the signed numerical range -2^{15} to $(+2^{15} - 1)$, or 8000_{16} to $7FFF_{16}$.

The instructions CMP, CMPB and CMPL do not affect the state of the V flag. Table 2–6 lists the instructions that are affected by V flag. See Chapter 6 for more details on instructions.

V can be summarized as follows:

- 0 V has been cleared.
- 1 An overflow has been detected, or V has been set.

| Instruction | Description |
|-------------------------|------------------------------|
| ABS ACC | if(ACC == 0x8000 0000) V = 1 |
| ABSTC ACC | if(ACC == 0x8000 0000) V = 1 |
| ADD ACC,#16bit << shift | V = 1 on signed overflow |
| ADD ACC,loc16 << shift | V = 1 on signed overflow |
| ADD ACC,loc16 << T | V = 1 on signed overflow |
| ADD AX,loc16 | V = 1 on signed overflow |
| ADD loc16,#16bitSigned | V = 1 on signed overflow |
| ADD loc16,AX | V = 1 on signed overflow |
| ADDB ACC,#8bit | V = 1 on signed overflow |
| ADDB AX,#8bitSigned | V = 1 on signed overflow |
| ADDCL ACC,loc32 | V = 1 on signed overflow |
| ADDCU ACC,loc16 | V = 1 on signed overflow |
| ADDL ACC,loc32 | V = 1 on signed overflow |
| ADDL loc32,ACC | V = 1 on signed overflow |
| ADDU ACC,loc16 | V = 1 on signed overflow |
| ADDUL ACC, loc32 | V = 1 on signed overflow |

Table 2–6. Instructions Affected by V flag

Central Processing Unit 2-21

| Instruction | Description | | | |
|---------------------------|-------------------------------|--|--|--|
| ADDUL P,loc32 | V = 1 on signed overflow | | | |
| B 16bitOff,COND | V = 0 if tested | | | |
| BF 16bitOff,COND | V = 0 if tested | | | |
| DEC loc16 | V = 1 on signed overflow | | | |
| DMAC ACC:P,loc32,*XAR7/++ | V = 1 on signed overflow | | | |
| IMACL P,loc32,*XAR7/++ | V = 1 on signed overflow | | | |
| IMPYAL P,XT,loc32 | V = 1 on signed overflow | | | |
| IMPYSL P,XT,loc32 | V = 1 on signed overflow | | | |
| INC loc16 | V = 1 on signed overflow | | | |
| MAC P,loc16,*XAR7/++ | V = 1 on signed overflow | | | |
| MAC P,loc16,0:pma | V = 1 on signed overflow | | | |
| MAX AX,loc16 | if((AX - [loc16]) > 0) V = 1 | | | |
| MAXL ACC,loc32 | if((ACC - [loc32]) > 0) V = 1 | | | |
| MIN AX,loc16 | if((AX - [loc16]) < 0) V = 1 | | | |
| MINL ACC, loc32 | if((ACC - [loc32]) < 0) V = 1 | | | |
| MOV loc16,AX,COND | V = 0 if tested | | | |
| MOVA T,loc16 | V = 1 on signed overflow | | | |
| MOVAD T,loc16 | V = 1 on signed overflow | | | |
| MOVB loc16,#8bit,COND | V = 0 if tested | | | |
| MOVL loc32,ACC,COND | V = 0 if tested | | | |
| MOVS T,loc16 | V = 1 on signed overflow | | | |
| MPYA P,loc16,#16bit | V = 1 on signed overflow | | | |
| MPYA P,T,loc16 | V = 1 on signed overflow | | | |
| MPYS P,T,loc16 | V = 1 on signed overflow | | | |
| NEG ACC | if(ACC == 0x8000 0000) V = 1 | | | |
| NEG AX | if(AX == 0x8000) V = 1 | | | |
| NEG64 ACC:P | if(ACC:P == 0x8000) V = 1 | | | |

Table 2–6. Instructions Affected by V flag (Continued)

| Instruction | Description | | | |
|-------------------------|-------------------------------|--|--|--|
| NEGTC ACC | if(TC == 1) | | | |
| | if(ACC == 0x8000 0000) V = 1 | | | |
| QMACL P,loc32,*XAR7/++ | V = 1 on signed overflow | | | |
| QMPYAL P,XT,loc32 | V = 1 on signed overflow | | | |
| QMPYSL P,XT,loc32 | V = 1 on signed overflow | | | |
| SAT ACC | if(OVC == 0) V = 0 else V = 1 | | | |
| SAT64 ACC:P | if(OVC == 0) V = 0 else V = 1 | | | |
| SB 8bitOff,COND | V = 0 if tested | | | |
| SBBU ACC,loc16 | V = 1 on signed overflow | | | |
| SQRA loc16 | V = 1 on signed overflow | | | |
| SQRS loc16 | V = 1 on signed overflow | | | |
| SUB ACC,#16bit << shift | V = 1 on signed overflow | | | |
| SUB ACC,loc16 << shift | V = 1 on signed overflow | | | |
| SUB ACC,loc16 << T | V = 1 on signed overflow | | | |
| SUB AX,loc16 | V = 1 on signed overflow | | | |
| SUB loc16,AX | V = 1 on signed overflow | | | |
| SUBB ACC,#8bit | V = 1 on signed overflow | | | |
| SUBBL ACC,loc32 | V = 1 on signed overflow | | | |
| SUBL ACC,loc32 | V = 1 on signed overflow | | | |
| SUBL loc32,ACC | V = 1 on signed overflow | | | |
| SUBR loc16,AX | V = 1 on signed overflow | | | |
| SUBRL loc32,ACC | V = 1 on signed overflow | | | |
| SUBU ACC,loc16 | V = 1 on signed overflow | | | |
| SUBUL ACC, loc32 | V = 1 on signed overflow | | | |
| SUBUL P,loc32 | V = 1 on signed overflow | | | |
| XB pma,COND | V = 0 if tested | | | |
| XCALL pma,COND | V = 0 if tested | | | |
| XMAC P,loc16,*(pma) | V = 1 on signed overflow | | | |

Table 2–6. Instructions Affected by V flag (Continued)

Central Processing Unit 2-23

Ν

| Instruction | Description |
|----------------------|--------------------------|
| XMACD P,loc16,*(pma) | V = 1 on signed overflow |
| XRETC COND | V = 0 if tested |

Table 2–6. Instructions Affected by V flag (Continued)

Negative flag. During certain operations, N is set if the result of the operation is a negative number or cleared if the result is a positive number. At reset, N is cleared. Bit 5

> Results in ACC are tested for the negative condition. Bit 31 of ACC is the sign bit. If bit 31 is a 0, ACC is positive; if bit 31 is a 1, ACC is negative. N is set if a result in ACC is negative or cleared if a result is positive.

> Results in AH, AL, and other 16-bit registers or data-memory locations are also tested for the negative condition. In these cases bit 15 of the value is the sign bit (1 indicates negative, 0 indicates positive). N is set if the value is negative or cleared if the value is positive.

> The TEST ACC instruction sets N if the value in ACC is negative. Otherwise the instruction clears N.

> As shown in Table 2–7, under overflow conditions, the way the N flag is set for compare operations is different from the way it is set for addition or subtraction operations. For addition or subtraction operations, the N flag is set to match the most significant bit of the truncated result. For compare operations, the N flag assumes infinite precision. This applies to operations whose result is loaded to ACC, AH, AL, another register, or a data-memory location.

A† B† (A - B) Subtraction Compare[‡] Neg (due to overflow in positive direction) Pos Neg N = 1 N = 0Pos Neg Pos (due to overflow in negative direction) N = 0N = 1 [†] For 32-bit data: Pos = Positive number from 0000 0000₁₆ to 7FFF FFFF₁₆ Neg = Negative number from 8000 000016 to FFFF FFFF16 For 16-bit data: Pos = Positive number from 0000_{16} to $7FF_{16}$

Table 2–7. Negative Flag Under Overflow Conditions

Neg = Negative number from 8000₁₆ to FFFF₁₆ [‡] The compare instructions are CMP, CMPB, CMPL, MIN, MAX, MINL, and MAXL.

N can be summarized as follows:

0 The tested number is positive, or N has been cleared.

1 The tested number is negative, or N has been set. ZZero flag. Z is set if the result of certain operations is 0 or is cleared if the result is nonzero.Bit 4This applies to results that are loaded into ACC, AH, AL, another register, or a data-memory
location. At reset, Z is cleared.

The TEST ACC instruction sets Z if the value in ACC is 0. Otherwise, it clears Z.

Z can be summarized as follows:

- 0 The tested number is nonzero, or Z has been cleared.
- 1 The tested number is 0, or Z has been set.

C Carry bit. This bit indicates when an addition or increment generates a carry or when a sub-Bit 3 traction, compare, or decrement generates a borrow. It is also affected by rotate operations on ACC and barrel shifts on ACC, AH, and AL.

During additions/increments, C is set if the addition generates a carry; otherwise C is cleared. There is one exception: If you are using the ADD instruction with a shift of 16, the ADD instruction can set C but cannot clear C.

During subtractions/decrements/compares, C is cleared if the subtraction generates a carry; otherwise C is set. There is one exception: if you are using the SUB instruction with a shift of 16, the SUB instruction can clear C but cannot set C.

This bit can be individually set and cleared by the SETC C instruction and CLRC C instruction, respectively. At reset, C is cleared.

C can be summarized as follows:

- 0 A subtraction generated a borrow, an addition did not generate a carry, or C has been cleared. *Exception:* An ADD instruction with a shift of 16 cannot clear C.
- 1 An addition generated a carry, a subtraction did not generate a borrow, or C has been set. *Exception:* A SUB instruction with a shift of 16 cannot set C.

Table 2–8 lists the bits that are affected by the C bit. For more information on instructions, see Chapter 6.

| Instruction | Affect of or Affect on C |
|-------------------------|-----------------------------|
| ABS ACC | C = 0 |
| ABSTC ACC | C = 0 |
| ADD ACC,#16bit << shift | C = 1 on carry else C = 0 |
| ADD ACC,loc16 << shift | if(shift == 16) |
| | C = 1 on carry |
| | if(shift != 16) |
| | C = 1 on carry else $C = 0$ |

Table 2–8. Bits Affected by the C Bit

| Instruction | Affect of or Affect on C |
|------------------------|--|
| ADD ACC,loc16 << T | C = 1 on carry else C = 0 |
| ADD AX,loc16 | C = 1 on carry else C = 0 |
| ADD loc16,#16bitSigned | C = 1 on carry else C = 0 |
| ADD loc16,AX | C = 1 on carry else C = 0 |
| ADDB ACC,#8bit | C = 1 on carry else C = 0 |
| ADDB AX,#8bitSigned | C = 1 on carry else C = 0 |
| ADDCL ACC, loc32 | ACC = ACC + [loc32] + C C = 1 on carry else C = 0 |
| ADDCU ACC,loc16 | ACC = ACC + [loc16] + C C = 1 on carry else C = 0 |
| ADDL ACC,loc32 | C = 1 on carry else C = 0 |
| ADDL loc32,ACC | C = 1 on carry else C = 0 |
| ADDU ACC,loc16 | C = 1 on carry else C = 0 |
| ADDUL ACC,loc32 | C = 1 on carry else C = 0 |
| ADDUL P,loc32 | C = 1 on carry else C = 0 |
| ASR AX,116 | C = AX(bit(shift-1)) |
| ASR AX,T | if(T == 0) C = 0 |
| | else C = AX(bit(T -1)) |
| ASR64 ACC:P,116 | C = P(bit(shift-1)) |
| ASR64 ACC:P,T | if(T == 0) C = 0 |
| | else C = P(bit(T-1)) |
| ASRL ACC,T | if(T == 0) C = 0 |
| | else C = ACC(bit(T-1)) |
| B 16bitOff,COND | C bit used as test condition |
| BF 16bitOff,COND | C bit used as test condition |
| CLRC C | C = 0 |
| CMP AX,loc16 | C = 0 on borrow else C = 1 |

Table 2–8. Bits Affected by the C Bit (Continued)

| Instruction | Affect of or Affect on C |
|---------------------------|------------------------------|
| CMP loc16,#16bitSigned | for([loc16] – 16bitSigned) |
| | C = 0 on borrow else $C = 1$ |
| CMPB AX,#8bit | C = 0 on borrow else $C = 1$ |
| CMPL ACC,loc32 | for(ACC – [loc32]) |
| | C = 0 on borrow else $C = 1$ |
| CMPL ACC,P << PM | for(ACC – P << PM) |
| | C = 0 on borrow else C = 1 |
| DEC loc16+ | C = 0 on borrow else $C = 1$ |
| DMAC ACC:P,loc32,*XAR7/++ | C = 1 on carry else C = 0 |
| IMACL P,loc32,*XAR7/++ | C = 1 on carry else C = 0 |
| IMPYAL P,XT,loc32 | C = 1 on carry else C = 0 |
| IMPYSL P,XT,loc32 | C = 0 on borrow else C = 1 |
| INC loc16 | C = 1 on carry else C = 0 |
| LSL ACC,116 | C = ACC(bit(32-shift)) |
| LSL ACC,T | if(T == 0) C = 0 |
| | else C = ACC(bit(32-T)) |
| LSL AX,116 | C = AX(bit(16-shift)) |
| LSL AX,T | if(T == 0) C = 0 |
| | else C = AX(bit(16–T)) |
| LSL64 ACC:P,116 | C = ACC(bit(32-shift)) |
| LSL64 ACC:P,T | if(T == 0) C = 0 |
| | else C = ACC(bit(32–T)) |
| LSLL ACC,T | if(T == 0) C = 0 |
| | else C = ACC(bit(32–T)) |
| LSR AX,116 | C = AX(bit(shift-1)) |
| LSR AX,T | if(T == 0) C = 0 |
| | else C = AX(bit(T-1)) |
| LSR64 ACC:P,116 | C = P(bit(shift-1)) |

Table 2–8. Bits Affected by the C Bit (Continued)

Central Processing Unit 2-27

| Instruction | Affect of or Affect on C |
|-----------------------|------------------------------|
| LSR64 ACC:P,T | if(T == 0) C = 0 |
| | else C = P(bit(T-1)) |
| LSRL ACC,T | if(T == 0) C = 0 |
| | else C = ACC(bit(T-1)) |
| MAC P,loc16,*XAR7/++ | C = 1 on carry else $C = 0$ |
| MAC P,loc16,0:pma | C = 1 on carry else $C = 0$ |
| MAX AX,loc16 | for(AX - [loc16]) |
| | C = 0 on borrow else $C = 1$ |
| MAXL ACC,loc32 | for(ACC – [loc32]) |
| | C = 0 on borrow else $C = 1$ |
| MIN AX,loc16 | for(AX – [loc16]) |
| | C = 0 on borrow else $C = 1$ |
| MINL ACC,loc32 | for(ACC – [loc32]) |
| | C = 0 on borrow else $C = 1$ |
| MOV loc16,AX,COND | C bit used as test condition |
| MOVA T,loc16 | C = 1 on carry else $C = 0$ |
| MOVAD T,loc16 | C = 1 on carry else $C = 0$ |
| MOVB loc16,#8bit,COND | C bit used as test condition |
| MOVL loc32,ACC,COND | C bit used as test condition |
| MOVS T,loc16 | C = 0 on borrow else C = 1 |
| MPYA P,loc16,#16bit | C = 1 on carry else $C = 0$ |
| MPYA P,T,loc16 | C = 1 on carry else $C = 0$ |
| MPYS P,T,loc16 | C = 0 on borrow else $C = 1$ |
| NEG ACC | if(ACC == 0) C = 1 |
| | else C = 0 |
| NEG AX | if(AX == 0) C = 1 |
| | else C = 0 |
| NEG64 ACC:P | if(ACC:P == 0) C = 1 |
| | else C = 0 |

Table 2–8. Bits Affected by the C Bit (Continued)

| Instruction | Affect of or Affect on C |
|-------------------------|------------------------------|
| NEGTC ACC | if(TC == 1) |
| | if(ACC == 0) C = 1 |
| | else C = 0 |
| QMACL P,loc32,*XAR7/++ | C = 1 on carry else $C = 0$ |
| QMPYAL P,XT,loc32 | C = 1 on carry else C = 0 |
| QMPYSL P,XT,loc32 | C = 0 on borrow else C = 1 |
| ROL ACC | C <- (ACC << 1) <- C(before) |
| ROR ACC | C(before) -> (ACC >> 1) -> C |
| SAT ACC | C = 0 |
| SAT64 ACC:P | C = 0 |
| SB 8bitOff,COND | C bit used as test condition |
| SBBU ACC,loc16 | ACC = ACC - ([loc16] + ~C) |
| | C = 0 on borrow else $C = 1$ |
| SETC C | C = 1 |
| SFR ACC,116 | C = ACC(bit(shift-1)) |
| SFR ACC,T | if(T == 0) C = 0 |
| | else C = ACC(bit(T-1)) |
| SQRA loc16 | C = 1 on carry else C = 0 |
| SQRS loc16 | C = 0 on borrow else C = 1 |
| SUB ACC,#16bit << shift | C = 0 on borrow else C = 1 |
| SUB ACC,loc16 << shift | if(shift == 16) |
| | C = 0 on borrow |
| | if(shift != 16) |
| | C = 0 on borrow else $C = 1$ |
| SUB ACC,loc16 << T | C = 0 on borrow else $C = 1$ |
| SUB AX,loc16 | C = 0 on borrow else C = 1 |
| SUB loc16,AX | C = 0 on borrow else C = 1 |
| SUBB ACC,#8bit | C = 0 on borrow else C = 1 |

Table 2–8. Bits Affected by the C Bit (Continued)

Central Processing Unit 2-29

| Instruction | Affect of or Affect on C |
|----------------------|----------------------------------|
| SUBBL ACC,loc32 | $ACC = ACC - ([loc32] + \sim C)$ |
| | C = 0 on borrow else $C = 1$ |
| SUBCU ACC,loc16 | for(ACC - [loc16]<<15) |
| | C = 0 on borrow else $C = 1$ |
| SUBCUL ACC,loc32 | for(ACC<<1 + P(31) - [loc32]) |
| | C = 0 on borrow else $C = 1$ |
| SUBL ACC,loc32 | C = 0 on borrow else $C = 1$ |
| SUBL loc32,ACC | C = 0 on borrow else C = 1 |
| SUBR loc16,AX | C = 0 on borrow else C = 1 |
| SUBRL loc32,ACC | C = 0 on borrow else $C = 1$ |
| SUBU ACC,loc16 | C = 0 on borrow else $C = 1$ |
| SUBUL ACC,loc32 | C = 0 on borrow else C = 1 |
| SUBUL P,loc32 | C = 0 on borrow else C = 1 |
| XB pma,COND | C bit used as test condition |
| XCALL pma,COND | C bit used as test condition |
| XMAC P,loc16,*(pma) | C = 1 on carry else C = 0 |
| XMACD P,loc16,*(pma) | C = 1 on carry else C = 0 |
| XRETC COND | C bit used as test condition |

Table 2–8. Bits Affected by the C Bit (Continued)

TC Bit 2 **Test/control flag.** This bit shows the result of a test performed by either the TBIT (test bit) instruction or the NORM (normalize) instruction.

The TBIT instruction tests a specified bit. When TBIT is executed, the TC bit is set if the tested bit is 1 or cleared if the tested bit is 0.

When a NORM instruction is executed, TC is modified as follows: If ACC holds 0, TC is set. If ACC does not hold 0, the CPU calculates the exclusive-OR of ACC bits 31 and 30, and then loads TC with the result.

This bit can be individually set and cleared by the SETC TC instruction and CLRC TC instruction, respectively. At reset, TC is cleared.

Table 2–9 lists the instructions that affect the TC bit. See the instruction set in Chapter 6 for a complete description of each instruction.

| Table 2–9. | Instructions | That Affect the TC Bi | t |
|------------|--------------|------------------------|---|
| 10010 L 0. | 1100100110 | That Thought the To Di | |

| Instruction | Affect on the TC bit |
|-----------------------|------------------------------------|
| ABSTC ACC | if(ACC < 0) TC = TC ^ 1 |
| B 16bitOff,COND | TC bit used as test condition |
| BF 16bitOff,COND | TC bit used as test condition |
| CLRC TC | TC = 0 |
| CMPR 0/1/2/3 | TC = 0 |
| | 0: if(AR(ARP) == AR0) TC = 1 |
| | 1: if(AR(ARP) < AR0) TC = 1 |
| | 2: if(AR(ARP) > AR0) TC = 1 |
| | 3: if(AR(ARP) != AR0) TC = 1 |
| CSB ACC | TC = N flag |
| MOV loc16,AX,COND | TC bit used as test condition |
| MOVB loc16,#8bit,COND | TC bit used as test condition |
| MOVL loc32,ACC,COND | TC bit used as test condition |
| NEGTC ACC | TC bit used as test condition |
| NORM ACC,XARn++/ | if(ACC = 0) |
| NORM ACC,*ind | TC = ACC(31) ^ ACC(30) |
| | else |
| | TC = 1 |
| SB 8bitOff,COND | TC bit used as test condi- tion |
| SBF 8bitOff,TC/NTC | TC bit used as test condition |
| SETC TC | TC = 1 |
| TBIT loc16,#bit | TC = [loc16(bit)] |
| TBIT loc16,T | TC = [loc16(15–T)] |
| TCLR loc16,#bit | TC = [loc16(bit)] |
| TSET loc16,#bit | TC = [loc16(bit)] |
| XB pma,COND | TC bit used as test condition |
| XCALL pma,COND | TC bit used as test condition |
| XRETC COND | TC bit used as test condition |

Central Processing Unit 2-31

| OVM Bit 1 | Overflow mode bit. When ACC accepts the result of an addition or subtraction and the result causes an overflow, OVM determines how the CPU handles the overflow as follows:. | | | |
|--------------|--|--|--|--|
| | 0 Results overflow nor scription for the OVC | nally in ACC. The OVC reflects the overflow (see the de- on page 2-16) | | |
| | 1 ACC is filled with eith | er its most positive or most negative value as follows: | | |
| | If ACC overflows in t ACC is then filled wit | וe positive direction (from 7FFF FFFF ₁₆ to 8000 0000 ₁₆), ז 7FFF FFFF ₁₆ . | | |
| | If ACC overflows in t ACC is then filled wit | ie negative direction (from 8000 0000 ₁₆ to 7FFF FFFF ₁₆), n 8000 0000 ₁₆ . | | |
| | This bit can be individually set and cleared by the SETC OVM instruction and CLRC OVM instruction, respectively. At reset, OVM is cleared. | | | |
| SXM Bit 0 | Sign-extension mode bit. SXM affects the MOV, ADD, and SUB instructions that use a 16-bit value in an operation on the 32-bit accumulator. When the 16-bit value is loaded into (MOV), added to (ADD), or subtracted from (SUB) the accumulator, SXM determines whether the value is sign extended during the operation as follows: | | | |
| | 0 Sign extension is sup | pressed. (The value is treated as unsigned.)i | | |
| | 1 Sign extension is ena | bled. (The value is treated as signed.) | | |
| | SXM also determines whether the accumulator is sign extended when it is shifted right by the SFR instruction. SXM does not affect instructions that shift the product register value; all right shifts of the product register value use sign extension. | | | |
| | his bit can be individually set | and cleared by the SETC SXM instruction and | | |

This bit can be individually set and cleared by the SETC SXM instruction and CLRC SXM instruction, respectively. At reset, SXM is cleared. Table 2–10 lists the instructions that are affected by SXM. See Chapter 6 for more details on instructions.

| Instruction | Description |
|-------------------------|-----------------|
| ADD ACC,#16bit << shift | Affected By SXM |
| ADD ACC,loc16 << shift | Affected By SXM |
| ADD ACC,loc16 << T | Affected By SXM |
| CLRC SXM | SXM = 0 |
| MOV ACC,#16bit << shift | Affected By SXM |
| MOV ACC,loc16 << shift | Affected By SXM |
| MOV ACC,loc16 << T | Affected By SXM |
| SETC SXM | SXM = 1 |
| SFR ACC,116 | Affected By SXM |
| SFR ACC,T | Affected By SXM |
| SUB ACC,#16bit << shift | Affected By SXM |
| SUB ACC,loc16 << shift | Affected By SXM |
| SUB ACC,loc16 << T | Affected By SXM |

Table 2–10. Instructions Affected by SXM

2.4 Status Register ST1

The following figure shows the bit fields of status register ST1. All of these bit fields are modified in the decode 2 phase of the pipeline. Detailed descriptions of these bits follow the figure.

| Figure 2–11. Bit Fields of Status Register 1 (S | ST1, | |
|---|------|--|
|---|------|--|

| 15 | | 13 | 12 | 11 | 10 | 9 | 8 |
|----------|---------|------|-------|---------|----------|---------|-------|
| ARP | | | XF | M0M1MAP | Reserved | OBJMODE | AMODE |
| | R/W-000 | | R/W-0 | R/W-1 | R/W-0 | R/W-0 | R/W-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IDLESTAT | EALLOW | LOOP | SPA | VMAP | PAGE0 | DBGM | INTM |
| R–0 | R/W-0 | R-0 | R/W-0 | R/W-1 | R/W-0 | R/W-1 | R/W-1 |

ARP

Bits 15-13

Auxiliary register pointer. This 3-bit field points to the current auxiliary register. This is one of the 32-bit auxiliary registers (XAR0–XAR7). The mapping of ARP values to auxiliary registers is as follows:

| ARP | ARP Selected Auxiliary Register | |
|-----|---------------------------------|--|
| 000 | XAR0 (selected at reset) | |
| 001 | XAR1 | |
| 010 | XAR2 | |
| 011 | XAR3 | |
| 100 | XAR4 | |
| 101 | XAR5 | |
| 110 | XAR6 | |
| 111 | XAR7 | |

XFXF status bit. This bit reflects the current state of the XFS output signal, which is compatible to the C2XLP CPU. This bit is set by the "SETC XF" instruction. This bit is
cleared by the "CLRC XF" instruction. The pipeline is not flushed when setting or clear-
ing this bit using the given instructions. This bit can be saved and restored by interrupts
and when restoring the ST1 register. This bit is set to 0 on reset.

MOM1MAPM0 and M1 mapping mode bit. The M0M1MAP bit should always remain set to 1 in
the C28x object mode. This is the default value at reset. The M0M1MAP bit may be set
low when operating in C27x-compatible mode. The effect of this bit, when low, is to
swap the location of blocks M0 and M1 only in program space and to set the stack
pointer default reset value to 0x000. C28x mode users should never set this bit to 0.

2-34

| Reserved Bit 10 | Reserved . This bit is reserved. Writes to this bit have no effect. |
|---------------------------|--|
| OBJMODE Bit 9 | Object compatibility mode bit . This mode is used to select between C27x object mode (OBJMODE == 0) and C28x object mode (OBJMODE == 1) compatibility. This bit is set by the "C28OBJ" (or "SETC OBJMODE") instructions. This bit is cleared by the "C27OBJ" (or "CLRC OBJMODE") instructions. The pipeline is flushed when setting or clearing this bit using the given instructions. This bit is saved and restored by interrupts and when restoring the ST1 register. This bit is set to 0 on reset. |
| AMODE Bit 8 | Address mode bit. This mode, in conjunction with the PAGE0 mode bit, is used to se- lect the appropriate addressing mode decodes. This bit is set by the "LPADDR" ("SETC AMODE") instructions. This bit is cleared by the "C28ADDR" (or "CLRC AMODE") in- structions. The pipeline is not flushed when setting or clearing this bit using the given instructions. This bit is saved and restored by interrupts and when restoring the ST1 register. This bit is set to 0 on reset. |
| | Note: Setting PAGE0 = AMODE = 1 will generate an illegal instruction trap ONLY for instructions that decode a memory or register addressing mode field (loc16 or loc32). |
| IDLESTAT Bit 7 | IDLE status bit. This ready-only bit is set when the IDLE instruction is executed. It is cleared by any one of the following events: |
| | An interrupt is serviced. |
| | An interrupt is not serviced but takes the CPU out of the IDLE state. |
| | A valid instruction enters the instruction register (the register that holds the instruction currently being decoded). |
| | A device reset occurs. |
| | When the CPU services an interrupt, the current value of IDLESTAT is saved on the stack (when ST1 is saved on the stack), and then IDLESTAT is cleared. Upon return from the interrupt, IDLESTAT is not restored from the stack. |
| EALLOW Bit 6 | Emulation access enable bit. This bit, when set, enables access to emulation and other protected registers. Set this bit by using the EALLOW instruction and clear this bit by using the EDIS instruction. See the data sheet for a particular device to determine the registers that are protected. |
| | When the CPU services an interrupt, the current value of EALLOW is saved on the stack (when ST1 is saved on the stack), and then EALLOW is cleared. Therefore, at the start of an interrupt service routine (ISR), access to protected registers is disabled. If the ISR must access protected registers, it must include an EALLOW instruction. At the end of the ISR, EALLOW can be restored by the IRET instruction. |
| LOOP Bit 5 | Loop instruction status bit. LOOP is set when a loop instruction (LOOPNZ or LOOPZ) reaches the decode 2 phase of the pipeline. The loop instruction does not end until a specified condition is met. When the condition is met, LOOP is cleared. LOOP is a read-only bit; it is not affected by any instruction except a loop instruction. |

| | When the CPU services an interrupt, the current value of LOOP is saved on the stack (when ST1 is saved on the stack), and then LOOP is cleared. Upon return from the interrupt, LOOP is not restored from the stack. |
|----------------------|--|
| SPA Bits 4 | Stack pointer alignment bit. SPA indicates whether the CPU has previously aligned the stack pointer to an even address by the ASP instruction: The stack pointer has not been aligned to an even address. The stack pointer has been aligned to an even address. |
| | When the ASP (align stack pointer) instruction is executed, if the stack pointer (SP) points to an odd address, SP is incremented by 1 so that it points to an even address, and SPA is set. If SP already points to an even address, SP is not changed, but SPA is cleared. When the NASP (unalign stack pointer) instruction is executed, if SPA is 1, SP is decremented by 1 and SPA is cleared. If SPA is 0, SP is not changed. |
| | At reset, SPA is cleared. |
| VMAP Bit 3 | Vector map bit. VMAP determines whether the CPU interrupt vectors (including the reset vector) are mapped to the lowest or highest addresses in program memory: 0 CPU interrupt vectors are mapped to the bottom of program memory, addresses 00 0000₁₆-00 003F₁₆. 1 CPU interrupt vectors are mapped to the top of program memory, addresses 3F FFC0₁₆-3F FFFF₁₆. |
| | On C28x designs, the VMAP signal is tied high internally, forcing the VMAP bit to be set high on a reset. |
| | This bit can be individually set and cleared by the SETC VMAP instruction and CLRC VMAP instruction, respectively. |
| PAGE0 Bit 2 | PAGE0 addressing mode configuration bit. PAGE0 selects between two mutually-exclusive addressing modes: PAGE0 direct addressing mode and PAGE0 stack addressing mode. Selection of the modes is as follows: PAGE0 stack addressing mode PAGE0 direct addressing mode |
| | Note: Illegal Instruction Trap |
| | Setting PAGE0 = AMODE = 1 will generate an illegal instruction trap. |
| | PAGE0 = 1 is included for compatibility with the C27x. the recommended operating mode for C28x is PAGE0 = 0. |
| | This bit can be individually set and cleared by the SETC PAGE0 instruction and CLRC PAGE0 instruction, respectively. At reset, the PAGE0 bit is cleared (PAGE0 stack addressing mode is selected). |
| | For details about the above addressing modes, see Chapter 5, Addressing Modes. |

| DBGI Bit 1 | M | Debug enable mask bit. When DBGM is set, the emulator cannot accesss memory or registers in real time. The debugger cannot update its windows. |
|----------------------|---|--|
| | | In the real-time emulation mode, if DBGM = 1, the CPU ignores halt requests or hard- ware breakpoints until DBGM is cleared. DBGM does not prevent the CPU from halting at a software breakpoint. One effect of this may be seen in real-time emulation mode. If you single-step an instruction in real time emulation mode and that instruction sets DBGM, the CPU continues to execute instructions until DBGM is cleared. |
| | | When you give the TI debugger the REALTIME command (to enter real-time mode), DBGM is forced to 0. Having DBGM = 0 ensures that debug and test direct memory accesses (DT-DMAs) are allowed; memory and register values can be passed to the host processor for updating debugger windows. |
| | | Before the CPU executes an interrupt service routine (ISR), it sets DBGM. When DBGM = 1, halt requests from the host processor and hardware breakpoints are ignored. If you want to single-step through or set breakpoints in a non-time-critical ISR, you must add a CLRC DBGM instruction at the beginning of the ISR. |
| | | DBGM is primarily used in emulation to block debug events in time-critical portions of program code. DBGM enables or disables debug events as follows: |
| | | 0 Debug events are enabled. |
| | | 1 Debug events are disabled. |
| | | When the CPU services an interrupt, the current value of DBGM is saved on the stack (when ST1 is saved on the stack), and then DBGM is set. Upon return from the inter- rupt, DBGM is restored from the stack. |
| | | This bit can be individually set and cleared by the SETC DBGM instruction and CLRC DBGM instruction, respectively. DBGM is also set automatically during interrupt operations. At reset, DBGM is set. Executing the ABORTI (abort interrupt) instruction also sets DBGM. |
| INTM Bit 0 | | Interrupt global mask bit. This bit globally enables or disables all maskable CPU interrupts (those that can be blocked by software): |
| | | 0 Maskable interrupts are globally enabled. To be acknowledged by the CPU, a maskable interrupt must also be locally enabled by the interrupt enable register (IER). |
| | | 1 Maskable interrupts are globally disabled. Even if a maskable interrupt is local- ly enabled by the IER, it is not acknowledged by the CPU. |
| | | INTM has no effect on the nonmaskable interrupts, including a hardware reset or the hardware interrupt $\overline{\text{NMI}}$. In addition, when the CPU is halted in real-time emulation mode, an interrupt enabled by the IER and the DBGIER will be serviced even if INTM is set to disable maskable interrupts. |

When the CPU services an interrupt, the current value of INTM is saved on the stack (when ST1 is saved on the stack), and then INTM is set. Upon return from the interrupt, INTM is restored from the stack.

This bit can be individually set and cleared by the SETC INTM instruction and CLRC INTM instruction, respectively. At reset, INTM is set. The value in INTM does not cause modification to the interrupt flag register (IFR), the interrupt enable register (IER), or the debug interrupt enable register (DBGIER).

2.5 Program Flow

The program control logic and program-address generation logic work together to provide proper program flow. Normally, the flow of a program is sequential: the CPU executes instructions at consecutive program-memory addresses. At times, a discontinuity is required; that is, a program must branch to a nonsequential address and then execute instructions sequentially at that new location. For this purpose, the '28x supports interrupts, branches, calls, returns, and repeats.

Proper program flow also requires smooth flow at the instruction level. To meet this need, the '28x has a protected pipeline and an instruction-fetch mechanism that attempts to keep the pipeline full.

2.5.1 Interrupts

Interrupts are hardware- or software-driven events that cause the CPU to suspend its current program sequence and execute a subroutine called an interrupt service routine. Interrupts are described in detail in Chapter 3.

2.5.2 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). Called subroutines or interrupt service routines are each concluded with a return instruction, which takes the return address from the stack or from XAR7 or RPC and places it into the program counter (PC).

The following branch instructions are conditional: B, BANZ, BAR, BF, SB, SBF, XBANZ, XCALL, and XRETC. They are executed only if a certain specified or predefined condition is met. For detailed descriptions of these instructions, see Chapter 6, *Assembly Language Instructions*.

2.5.3 Repeating a Single Instruction

The repeat (RPT) instruction allows the execution of a single instruction (N + 1) times, where N is specified as an operand of the RPT instruction. The instruction is executed once and then repeated N times. 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 0. For a description of RPT and a list of repeatable instructions, see Chapter 6, *Assembly Language Instructions*.

2.5.4 Instruction Pipeline

Each instruction passes through eight independent phases that form an instruction pipeline. At any given time, up to eight instructions may be active, each in a different phase of completion. Not all reads and writes happen in the same phases, but a pipeline-protection mechanism stalls instructions as needed to ensure that reads and writes to the same location happen in the order in which they are programmed.

To maximize pipeline efficiency, an instruction-fetch mechanism attempts to keep the pipeline full. Its role is to fill an instruction-fetch queue, which holds instructions in preparation for decoding and execution. The instruction-fetch mechanism fetches 32-bits at a time from program memory; it fetches one 32-bit instruction or two 16-bit instructions.

The instruction-fetch mechanism uses three program-address counters: the program counter (PC), the instruction counter (IC), and the fetch counter (FC). When the pipeline is full the PC will always point to the instruction in its decode 2 pipeline phase. The IC points to the next instruction to be processed. When the PC points to a 1-word instruction, IC = (PC+1); when the PC points to a 2-word instruction, IC = (PC+2). The value in the FC is the address from which the next fetch is to be made.

The pipeline and the instruction-fetch mechanism are described in more detail in Chapter 4, *Pipeline*.

2.6 Multiply Operations

The C28x features a hardware multiplier that can perform 16-bit X 16-bit or 32-bit X 32-bit fixed-point multiplication. This functionality is enhanced by 16-bit X 16-bit multiply and accumulate (MAC), 32 X 32 MAC, and 16-bit X 16-bit dual MAC (DMAC) instructions. This section describes the components involved in each type of multiplication.

2.6.1 16-bit X 16-bit Multiplication

The C28x multiplier can perform a 16-bit X 16-bit multiplication to produce a signed or unsigned 32-bit product. Figure 2–12 shows the CPU components involved in this multiplication.

The multiplier accepts two 16-bit inputs:

- One input is from the upper 16 bits of the multiplicand register (T). Most 16 X 16 multiplication instructions require that you load T from a datamemory location or a register before you execute the instruction. However, the MAC and some versions of the MPY and MPYA instructions load T for you before the multiplication.
- The other input is from one of the following:
 - A data-memory location or a register (depending on which you specify in the multiply instruction).
 - An instruction opcode. Some C28x multiply instructions allow you to include a constant as an operation.

After the value has been multiplied by the second value, the 32-bit result is stored in one of two places, depending on the particular multiply instruction: the 32-bit product register (P) or the 32-bit accumulator (ACC).

One special 16-bit X 16-bit multiplication instruction takes two 32-bit input values as its operands. This instruction is the 16 X 16 DMAC instruction, which performs dual 16 X 16 MAC operations in one instruction. In this case, the ACC contains the result of multiplying and adding the upper word of the 32-bit operands. The P register contains the result of multiplying and adding the results of the lower word of the 32-bit operands.



Figure 2–12. Conceptual Diagram of Components Involved in 16 X16-Bit Multiplication

2.6.2 32-Bit X 32-Bit Multiplication

The C28x multiplier can also perform 32-bit by 32-bit multiplication. Figure 2–13 shows the CPU components involved n this multiplication. In this case, the multiplier accepts two 32-bit inputs:

The first input is from one of the following:

- A program memory location. Some C28x 32 X 32 multiply MAC-type instructions such as IMACL and QMACL take one data value directly from memory using the program-address bus.
- The 32-bit multiplicand register (XT). Most 32 X 32-bit multiplication instructions require that you load XT from data memory or a register before you execute the instruction.
- A data-memory location or a register (depending on which you specify in the multiply instruction).

After the two values have ben multiplied, 32 bits of the 64-bit result are stored in the product register (P). You can control which half is stored (upper 32 bits or lower 32 Bits) and whether the multiplication is signed or unsigned by the instruction used.
If you need support for larger data values, the 32 X 32 multiplication instructions can be combined to implement $32 \times 32 = 64$ -bit or $64 \times 64 = 128$ -bit math.

Figure 2–13. Conceptual Diagram of Components Involved in 32 X 32-Bit Multiplication



2.7 Shift Operations

The shifter holds 64 bits and accepts either a 16-bit, 32-bit, or 64-bit input value. When the input value has 16 bits, the value is loaded into the 16 least significant bits (LSBs) of the shifter. When the input value has 32 bits, the value is loaded into the 32 LSBs of the shifter. Depending on the instruction that uses the shifter, the output of the shifter may be all of its 64 bits or just its 16 LSBs.

When a value is shifted *right* by an amount N, the N LSBs of the value are lost and the bits to the left of the value are filled with all 0s or all 1s. If sign extension is specified, the bits to the left are filled with copies of the sign bit. If sign extension is not specified, the bits to the left are filled with 0s, or zero filled.

When a value is shifted *left* by an amount N, the bits to the right of the shifted value are zero filled. If the value has 16 bits and sign extension is specified, the bits to the left are filled with copies of the sign bit. If the value has 16 bits and sign extension is not specified, the bits to the left are zero filled. If the value has 32 bits, the N MSBs of the value are lost, and sign extension is irrelevant.

Table 2–11 lists the instructions that use the shifter and provides an illustration of the corresponding shifter operation. The table uses the following graphical symbols:

| Shift left | This symbol represents the 32-bit shifter. The text inside the box indicates the direction of the shift. |
|---------------|---|
| 0 | This symbol indicates zero filling. |
| Sign | This symbol indicates sign extending. |
| SXM 0/Sign | This symbol indicates that the MSBs of the shifter depend on the sign-extension mode bit (SXM). If $SXM = 0$, the MSBs are zero filled after the shift. If $SXM = 1$, the MSBs are filled with the sign of the shifted value. |
| с | This symbol indicates the carry bit (C). |

For explanations of the instruction syntaxes listed in Table 2–11, see Chapter 6, *Assembly Language Instructions*.

Table 2–11. Shift Operations



Table 2–11. Shift Operations (Continued)



Table 2–11. Shift Operations (Continued)





Table 2–11. Shift Operations (Continued)



Central Processing Unit 2-49

This page intentionally left blank.

This page intentionally left blank.

Chapter 3

CPU Interrupts and Reset

This chapter describes the available CPU interrupts and how they are handled by the CPU. It also explains how to control those interrupts that can be controlled through software. Finally, it describes how a hardware reset affects the CPU.

| Торі | C Page |
|------|---|
| 3.1 | CPU Interrupts Overview 3-2 |
| 3.2 | CPU Interrupt Vectors and Priorities |
| 3.3 | Maskable Interrupts: INT1-INT14, DLOGINT, and RTOSINT 3-6 |
| 3.4 | Standard Operation for Maskable Interrupts |
| 3.5 | Nonmaskable Interrupts 3-17 |
| 3.6 | Illegal-Instruction Trap 3-22 |
| 3.7 | Hardware Reset (RS) 3-23 |
| | |

3.1 CPU Interrupts Overview

Interrupts are hardware- or software-driven signals that cause the C28x CPU to suspend its current program sequence and execute a subroutine. Typically, interrupts are generated by peripherals or hardware devices that need to give data to or take data from the C28x (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).

On the C28x, interrupts can be triggered by software (the INTR, OR IFR, or TRAP instruction) or by hardware (a pin, an external peripheral, or on-chip peripheral/logic). If hardware interrupts are triggered at the same time, the C28x services them according to a set priority ranking.

Some 28x devices include a peripheral interrupt expansion (PIE) module that multiplexes interrupts from a number of peripherals into a single CPU interrupt. The PIE module provides additional control before an interrupt reaches the C28x CPU. See the *TMS320C8x System and Interrupts Reference Guide* (literature number SPRU078) for more details.

At the CPU level, each of the C28x interrupts, whether hardware or software, can be placed in one of the following two categories:

- Maskable interrupts. These are interrupts that can be blocked (masked) or enabled (unmasked) through software.
- Nonmaskable interrupts. These interrupts cannot be blocked. The C28x will immediately approve this type of interrupt and branch to the corresponding subroutine. All software-initiated interrupts are in this category.

The C28x handles interrupts in four main phases:

- Receive the interrupt request. Suspension of the current program sequence must be requested by a software interrupt (from program code) or a hardware interrupt (from a pin or an on-chip device).
- Approve the interrupt. The C28x must approve the interrupt request. If the interrupt is maskable, certain conditions must be met in order for the C28x to approve it. For nonmaskable hardware interrupts and for software interrupts, approval is immediate.
- 3) **Prepare for the interrupt service routine and save register values.** The main tasks performed in this phase are:
 - Complete execution of the current instruction and flush from the pipeline any instructions that have not reached the decode 2 phase.
 - Automatically save most of the current program context by saving the following registers to the stack: ST0, T, AL, AH, PL, PH, AR0, AR1, DP, ST1, DBGSTAT, PC, and IER.

- Fetch the interrupt vector and load it into the program counter (PC). For devices with a PIE module, the vector fetched will depend on the setting of the PIE enable and flag registers.
- 4) Execute the interrupt service routine. The C28x branches to its corresponding subroutine called an interrupt service routine (ISR). The C28x branches to the address (vector) you store at a predetermined vector location and executes the ISR you have written.

3.2 CPU Interrupt Vectors and Priorities

The C28x supports 32 CPU interrupt vectors, including the reset vector. Each vector is a 22-bit address that is the start address for the corresponding interrupt service routine (ISR). Each vector is stored in 32 bits at two consecutive addresses. The location at the lower address holds the 16 least significant bits (LSBs) of the vector. The location at the higher address holds the 6 most significant bits (MSBs) right-justified. When an interrupt is approved, the 22-bit vector is fetched, and the 10 MSBs at the higher address are ignored.

For devices with a PIE module, this table is re-mapped and expanded into the PIE vector table.

Table 3–1 lists the available CPU interrupt vectors and their locations. The addresses are shown in hexadecimal form. The table also shows the priority of each of the hardware interrupts.

| | Absolute Addre | ss (hexadecimal) | Hardware | |
|--------|----------------|------------------------------|-------------|-----------------------|
| Vector | VMAP = 0 | VMAP = 1 [†] | Priority | Description |
| RESET | 00 0000 | 3F FFC0 | 1 (highest) | Reset |
| INT1 | 00 0002 | 3F FFC2 | 5 | Maskable interrupt 1 |
| INT2 | 00 0004 | 3F FFC4 | 6 | Maskable interrupt 2 |
| INT3 | 00 0006 | 3F FFC6 | 7 | Maskable interrupt 3 |
| INT4 | 00 0008 | 3F FFC8 | 8 | Maskable interrupt 4 |
| INT5 | 00 000A | 3F FFCA | 9 | Maskable interrupt 5 |
| INT6 | 00 000C | 3F FFCC | 10 | Maskable interrupt 6 |
| INT7 | 00 000E | 3F FFCE | 11 | Maskable interrupt 7 |
| INT8 | 00 0010 | 3F FFD0 | 12 | Maskable interrupt 8 |
| INT9 | 00 0012 | 3F FFD2 | 13 | Maskable interrupt 9 |
| INT10 | 00 0014 | 3F FFD4 | 14 | Maskable interrupt 10 |
| INT11 | 00 0016 | 3F FFD6 | 15 | Maskable interrupt 11 |
| INT12 | 00 0018 | 3F FFD8 | 16 | Maskable interrupt 12 |
| INT13 | 00 001A | 3F FFDA | 17 | Maskable interrupt 13 |
| INT14 | 00 001C | 3F FFDC | 18 | Maskable interrupt 14 |

Table 3–1. Interrupt Vectors and Priorities

[†] For C28x catalog devices, VMAP = 1 at reset.

[‡] Interrupts DLOGINT and RTOSINT are generated by the emulation logic internal to the CPU.

| | Absolute Addre | ss (hexadecimal) | Hardware | |
|----------------------|----------------|------------------------------|-------------|---|
| Vector | VMAP = 0 | VMAP = 1 [†] | Priority | Description |
| DLOGINT‡ | 00 001E | 3F FFDE | 19 (lowest) | Maskable data log interrupt |
| RTOSINT [‡] | 00 0020 | 3F FFE0 | 4 | Maskable real-time operating system interrupt |
| Reserved | 00 0022 | 3F FFE2 | 2 | Reserved |
| NMI | 00 0024 | 3F FFE4 | 3 | Nonmaskable interrupt |
| ILLEGAL | 00 0026 | 3F FFE6 | - | Illegal-instruction trap |
| USER1 | 00 0028 | 3F FFE8 | - | User-defined software interrupt |
| USER2 | 00 002A | 3F FFEA | - | User-defined software interrupt |
| USER3 | 00 002C | 3F FFEC | - | User-defined software interrupt |
| USER4 | 00 002E | 3F FFEE | - | User-defined software interrupt |
| USER5 | 00 0030 | 3F FFF0 | - | User-defined software interrupt |
| USER6 | 00 0032 | 3F FFF2 | - | User-defined software interrupt |
| USER7 | 00 0034 | 3F FFF4 | - | User-defined software interrupt |
| USER8 | 00 0036 | 3F FFF6 | - | User-defined software interrupt |
| USER9 | 00 0038 | 3F FFF8 | - | User-defined software interrupt |
| USER10 | 00 003A | 3F FFFA | - | User-defined software interrupt |
| USER11 | 00 003C | 3F FFFC | - | User-defined software interrupt |
| USER12 | 00 003E | 3F FFFE | - | User-defined software interrupt |

Table 3–1. Interrupt Vectors and Priorities (Continued)

[†] For C28x catalog devices, VMAP = 1 at reset.

[‡] Interrupts DLOGINT and RTOSINT are generated by the emulation logic internal to the CPU.

The vector table can be mapped to the top or bottom of program space, depending on the value of the vector map bit (VMAP) in status register ST1. (ST1 is described in section 2.4 on page 2-34.) If the VMAP bit is 0, the vectors are mapped beginning at address 00 0000₁₆. If the VMAP bit is 1, the vectors are mapped beginning at address 3F FFC0₁₆. Table 3–1 lists the absolute addresses for VMAP = 0 and VMAP = 1.

The VMAP bit can be set by the SETC VMAP instruction and cleared by the CLRC VMAP instruction. The reset value of VMAP is 1.

3.3 Maskable Interrupts: INT1–INT14, DLOGINT, and RTOSINT

INT1-INT14 are 14 general-purpose interrupts. DLOGINT (the data log interrupt) and RTOSINT (the real-time operating system interrupt) are available for emulation purposes. These interrupts are supported by three dedicated registers: the CPU interrupt flag register (IFR), the CPU interrupt enable register (IER), and the CPU debug interrupt enable register (DBGIER).

The 16-bit IFR contains flag bits that indicate which of the corresponding interrupts are pending (waiting for approval from the CPU). The external input lines INT1–INT14 are sampled at every CPU clock cycle. If an interrupt signal is recognized, the corresponding bit in the IFR is set and latched. For DLOGINT or RTOSINT, a signal sent by the CPU on-chip analysis logic causes the corresponding flag bit to be set and latched. You can set one or more of the IFR bits at the same time by using the OR IFR instruction. More details about the IFR are given in section 3.3.1. The on-chip analysis resources are introduced in Chapter 7.

The interrupt enable register (IER) and the debug interrupt enable register (DBGIER) each contain bits for individually enabling or disabling the maskable interrupts. To enable one of the interrupts in the IER, you set the corresponding bit in the IER; to enable the same interrupt in the DBGIER, you set the corresponding bit in the DBGIER. The DBGIER indicates which interrupts can be serviced when the CPU is in the real-time emulation mode. The IER and the DBGIER are discussed more in section 3.3.2. Real-time mode is discussed in section 7.4.2 on page 7-9.

The maskable interrupts also share bit 0 in status register ST1. This bit, the interrupt global mask bit (INTM), is used to globally enable or globally disable these interrupts. When INTM = 0, these interrupts are globally enabled. When INTM = 1, these interrupts are globally disabled. You can set and clear INTM with the SETC INTM and CLRC INTM instructions, respectively. ST1 is described in section 2.4 on page 2-34.

After a flag has been latched in the IFR, the corresponding interrupt is not serviced until it is appropriately enabled by two of the following: the IER, the DBGIER, and the INTM bit. As shown in Table 3–2, the requirements for enabling the maskable interrupts depend on the interrupt-handling process used. In the standard process, which occurs in most circumstances, the DBGIER is ignored. When the C28x is in real-time emulation mode and the CPU is halted, a different process is used. In this special case, the DBGIER is used and the INTM bit is ignored. (If the DSP is in real-time mode and the CPU is running, the standard interrupt-handling process applies.) Once an interrupt has been requested and properly enabled, the CPU prepares for and then executes the corresponding interrupt service routine. For a detailed description of this process, see section 3.4.

Table 3–2. Requirements for Enabling a Maskable Interrupt

| Interrupt-Handling Process | Interrupt Enabled If |
|--------------------------------------|--|
| Standard | INTM = 0 and bit in IER is 1 |
| DSP in real-time mode and CPU halted | Bit in IER is 1 and bit in DBGIER is 1 |

As an example of varying interrupt-enable requirements, suppose you want interrupt $\overline{INT5}$ enabled. This corresponds to bit 4 in the IER and bit 4 in the DBGIER. Usually, $\overline{INT5}$ is enabled if INTM = 0 and IER(4) = 1. In real-time emulation mode with the CPU halted, $\overline{INT5}$ is enabled if IER(4) = 1 and DBGIER(4) = 1.

3.3.1 CPU Interrupt Flag Register (IFR)

Figure 3–1 shows the IFR. If a maskable interrupt is pending (waiting for approval from the CPU), the corresponding IFR bit is 1; otherwise, the IFR bit is 0. To identify pending interrupts, use the PUSH IFR instruction and then test the value on the stack. Use the OR IFR instruction to set IFR bits, and use the AND IFR instruction to clear pending interrupts. When a hardware interrupt is serviced, or when an INTR instruction is executed, the corresponding IFR bit is cleared. All pending interrupts are cleared by the AND IFR, #0 instruction or by a hardware reset.

Notes:

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

Figure 3–1. Interrupt Flag Register (IFR)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---------|---------|-------|-------|-------|-------|-------|-------|
| RTOSINT | DLOGINT | INT14 | INT13 | INT12 | INT11 | INT10 | INT9 |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| INT8 | INT7 | INT6 | INT5 | INT4 | INT3 | INT2 | INT1 |
| 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: R = Read access; W = Write access; value following dash (-) is value after reset.

Bits 15 and 14 of the IFR correspond to the interrupts RTOSINT and DLOGINT:

| RTOSINT | Real-time opera | ating system interrupt flag | |
|---------|----------------------------|--|--|
| Bit 15 | RTOSINT = 0 | RTOSINT is not pending. | |
| | RTOSINT = 1 | RTOSINT is pending. | |
| DLOGINT | Data log interrupt flag | | |
| | | | |
| Bit 14 | DLOGINT = 0 | DLOGINT is not pending. | |
| Bit 14 | DLOGINT = 0 DLOGINT = 1 | DLOGINT is not pending. DLOGINT is pending. | |

For bits INT1-INT14, the following general description applies:

| INTx | Interrupt x fla | ag (x = 1, 2, 3,, or 14) |
|-----------|-----------------|--------------------------|
| Bit (x−1) | INTx = 0 | INTx is not pending. |
| | INTx = 1 | INTx is pending. |

3.3.2 CPU Interrupt Enable Register (IER) and CPU Debug Interrupt Enable Register (DBGIER)

Figure 3–2 shows the IER. To enable an interrupt, set its corresponding bit to 1. To disable an interrupt, clear its corresponding bit to 0. Two syntaxes of the MOV instruction allow you to read from the IER and write to the IER. In addition, the OR IER instruction enables you to set IER bits, and the AND IER instruction enables you to clear IER bits. When a hardware interrupt is serviced, or when an INTR instruction is executed, the corresponding IER bit is cleared. At reset, all the IER bits are cleared to 0, disabling all the corresponding interrupts.

Note:

When an interrupt is requested by the TRAP instruction, if the corresponding IER bit is set, the CPU does not clear it automatically. If an application requires that the IER bit be cleared, the bit must be cleared in the interrupt service routine.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---------|---------|-------|-------|-------|-------|-------|-------|
| | RTOSINT | DLOGINT | INT14 | INT13 | INT12 | INT11 | INT10 | INT9 |
| | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
| _ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ĺ | INT8 | INT7 | INT6 | INT5 | INT4 | INT3 | INT2 | INT1 |
| | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

Figure 3–2. Interrupt Enable Register (IER)

Note: R = Read access; W = Write access; value following dash (-) is value after reset.

Note:

When using the AND IER and OR IER instructions, make sure that they do not modify the state of bit 15 (RTOSINT) unless a real-time operating system is present.

Bits 15 and 14 of the IER enable or disable the interrupts RTOSINT and DLOGINT:

| RTOSINT | Real-time oper | ating system interrupt enable bit |
|---------|------------------|-----------------------------------|
| Bit 15 | RTOSINT = 0 | RTOSINT is disabled. |
| | RTOSINT = 1 | RTOSINT is enabled. |
| DLOGINT | Data log interro | upt enable bit |
| Bit 14 | DLOGINT = 0 | DLOGINT is disabled. |
| | | |

For bits INT1–INT14, the following general description applies:

| INTx | Interrupt x e | nable bit (x = 1, 2, 3,, or 14) |
|-----------|---------------|-----------------------------------|
| Bit (x−1) | INTx = 0 | INTx is disabled. |
| | INTx = 1 | $\overline{\rm INTx}$ is enabled. |

Figure 3–3 shows the DBGIER, which is used only when the CPU is halted in real-time emulation mode. An interrupt enabled in the DBGIER is defined as a *time-critical interrupt*. When the CPU is halted in real-time mode, the only interrupts that are serviced are time-critical interrupts that are also enabled in the IER. If the CPU is running in real-time emulation mode, the standard interrupt-handling process is used and the DBGIER is ignored.

As with the IER, you can read the DBGIER to identify enabled or disabled interrupts and write to the DBGIER to enable or disable interrupts. To enable an interrupt, set its corresponding bit to 1. To disable an interrupt, set its corresponding bit to 0. Use the PUSH DBGIER instruction to read from the DBGIER and the POP DBGIER instruction to write to the DBGIER. At reset, all the DBGIER bits are set to 0.

8

INT9

R/W-0

0

INT1

R/W-0

INT2

R/W-0

15 14 13 12 11 10 9 RTOSINT DLOGINT INT14 INT13 INT12 INT11 INT10 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 7 6 5 4 3 2 1

INT5

R/W-0

Figure 3–3. Debug Interrupt Enable Register (DBGIER)

Note: R = Read access; W = Write access; value following dash (-) is value after reset.

INT6

R/W-0

INT7

R/W-0

Bits 15 and 14 of the DBGIER enable or disable the interrupts RTOSINT and DLOGINT:

INT3

R/W-0

INT4

R/W-0

For bits INT1–INT14, the following general description applies:

| INTx | Interrupt x debug enable bit (x = 1, 2, 3,, or 14) | |
|-----------|--|-------------------|
| Bit (x−1) | INTx = 0 | INTx is disabled. |
| | INTx = 1 | INTx is enabled. |

INT8

R/W-0

3.4 Standard Operation for Maskable Interrupts

The flow chart in Figure 3–4 shows the standard process for handling interrupts. Section 7.4.2 on page 7-9 contains information on handling interrupts when the DSP is in real-time mode and the CPU is halted. When more than one interrupt is requested at the same time, the C28x services them one after another according to their set priority ranking. See the priorities in Table 3–1 on page 3-4.

Figure 3–4 is not meant to be an exact representation of how an interrupt is handled. It is a conceptual model of the important events.



Figure 3–4. Standard Operation for CPU Maskable Interrupts

What following list explains the steps shown in Figure 3-4:

- 1) Interrupt request sent to CPU. One of the following events occurs:
 - One of the pins INT1-INT14 is driven low by an external event, peripheral or PIE interrupt request..
 - The CPU emulation logic sends to the CPU a signal for DLOGINT or RTOSINT.
 - One of the interrupts INT1-INT14, DLOGINT, and RTOSINT is initiated by way of the OR IFR instruction.
- 2) Set corresponding IFR flag bit. When the CPU detects a valid interrupt in step 1, it sets and latches the corresponding flag in the interrupt flag register (IFR). This flag stays latched even if the interrupt is not approved by the CPU in step 3. The IFR is explained in detail in section 3.3.1.
- 3) Is the interrupt enabled in IER? Is the interrupt enabled by INTM bit? The CPU approves the interrupt only if the following conditions are true:
 - The corresponding bit in the IER is 1.
 - The INTM bit in ST1 is 0.

Once an interrupt has been enabled and then approved by the CPU, no other interrupts can be serviced until the CPU has begun executing the interrupt service routine for the approved interrupt (step 13). The IER is described in section 3.3.2. ST1 is described in section 2.4 on page 2-34.

- 4) Clear corresponding IFR bit. Immediately after the interrupt is approved, its IFR bit is cleared. If the interrupt signal is kept low, the IFR register bit will be set again. However, the interrupt is not immediately serviced again. The CPU blocks new hardware interrupts until the interrupt service routine (ISR) begins. In addition, the IER bit is cleared (in step 10) before the ISR begins; therefore, an interrupt from the same source cannot disturb the ISR until the IER bit is set again by the ISR.
- 5) **Empty the pipeline.** The CPU completes any instructions that have reached or passed their decode 2 phase in the instruction pipeline. Any instructions that have not reached this phase are flushed from the pipeline.
- 6) **Increment and temporarily store PC.** The PC is incremented by 1 or 2, depending on the size of the current instruction. The result is the *return address*, which is temporarily saved in an internal hold register. During the automatic context save (step 9), the return address is pushed onto the stack.

- 7) Fetch interrupt vector. The PC is filled with the address of the appropriate interrupt vector, and the vector is fetched from that location. To determine which vector address has been assigned to each of the interrupts, see section 3.2, *Interrupt Vectors*, on page 3-4 or, if your device uses a PIE module, see the System and Interrupts Reference Guide for your specific device.
- 8) Increment SP by 1. The stack pointer (SP) is incremented by 1 in preparation for the automatic context save (step 9). During the automatic context save, the CPU performs 32-bit accesses, and the CPU expects 32-bit accesses to be aligned to even addresses by the memory wrapper. Incrementing SP by 1 ensures that the first 32-bit access does not overwrite the previous stack value.
- 9) Perform automatic context save. A number of register values are saved automatically to the stack. These registers are saved in pairs; each pair is saved in a single 32-bit operation. At the end of each 32-bit save operation, the SP is incremented by 2. Table 3–3 shows the register pairs and the order in which they are saved. The CPU expects all 32-bit saves to be even-word aligned by the memory wrapper. As shown in the table, the SP is not affected by this alignment.

| Save Register | | Bit 0 of Storage Address | | |
|------------------------|-----------------|--|--|--|
| Operation [†] | Pairs | SP Starts at Odd Address | SP Starts at Even Address | |
| | | $1 \leftarrow SP$ position before step 8 | 1 | |
| 1st | ST0 | 0 | $0 \leftarrow SP \text{ position before step 8}$ | |
| | т | 1 | 1 | |
| 2nd | AL | 0 | 0 | |
| | AH | 1 | 1 | |
| 3rd | PL [‡] | 0 | 0 | |
| | PH | 1 | 1 | |
| 4th | AR0 | 0 | 0 | |
| | AR1 | 1 | 1 | |
| 5th | ST1 | 0 | 0 | |
| | DP | 1 | 1 | |

Table 3–3. Register Pairs Saved and SP Positions for Context Saves

| Save | Register Pairs | Bit 0 of Storage Address | | |
|------------------------|-------------------------------|---------------------------------------|---------------------------------------|--|
| Operation [†] | | SP Starts at Odd Address | SP Starts at Even Address | |
| 6th | IER | 0 | 0 | |
| | DBGSTAT§ | 1 | 1 | |
| 7th | Return address (low half) | 0 | 0 | |
| | Return address (high half) | 1 | 1 | |
| | | $0 \leftarrow SP$ position after save | 0 | |
| | | 1 | $1 \leftarrow SP$ position after save | |

Table 3–3. Register Pairs Saved and SP Positions for Context Saves (Continued)

[†] All registers are saved as pairs, as shown.

[‡] The P register is saved with 0 shift (CPU ignores current state of the product shift mode bits, PM, in status register 0).

§ The DBGSTAT register contains special emulation information.

- 10) **Clear corresponding IER bit.** After the IER register is saved on the stack in step 9, the CPU clears the IER bit that corresponds to the interrupt being handled. This prevents reentry into the same interrupt. If you want to nest occurrences of the interrupt, have the ISR set that IER bit again.
- 11) Set INTM and DBGM. Clear LOOP, EALLOW, and IDLESTAT. All these bits are in status register ST1. By setting INTM to 1, the CPU prevents maskable interrupts from disturbing the ISR. If you wish to nest interrupts, have the ISR clear the INTM bit. By setting DBGM to 1, the CPU prevents debug events from disturbing time-critical code in the ISR. If you do not want debug events blocked, have the ISR clear DBGM.

The CPU clears LOOP, EALLOW, and IDLESTAT so that the ISR operates within a new context.

- 12) **Load PC with fetched vector.** The PC is loaded with the interrupt vector that was fetched in step 7. The vector forces program control to the ISR.
- 13) **Execute interrupt service routine.** Here is where the CPU executes the program code you have prepared to handle the interrupt. A typical ISR is shown in Example 3–1.

Although a number of register values are saved automatically in step 10, if the ISR uses other registers, you may need to save the contents of these registers at the beginning of the ISR. These values must then be restored before the return from the ISR. The ISR in Example 3–1 saves and restores auxiliary registers AR1H:AR0H, XAR2–XAR7, and the temporary register XT. If you want the ISR to inform a peripheral that the interrupt is being serviced, you can use the IACK instruction to send an interrupt acknowledge signal. The IACK instruction accepts a 16-bit constant as an operand. For a detailed description of the IACK instruction, see Chapter 6, *C28x Assembly Language Instructions*.

14) **Program continues.** If the interrupt is not approved by the CPU, the interrupt is ignored, and the program continues uninterrupted. If the interrupt is approved, its interrupt service routine is executed and the program continues where it left off (at the return address).

Example 3–1. Typical ISR

C28x Full Context Save/Restore

| INTX: | .; | 8 cycles | | |
|-------|------------------|------------|---|--------|
| | PUSH | AR1H:AR0H | ; | 32-bit |
| | PUSH | XAR2 | ; | 32-bit |
| | PUSH | XAR3 | ; | 32-bit |
| | PUSH | XAR4 | ; | 32-bit |
| | PUSH | XAR5 | ; | 32-bit |
| | PUSH | XAR6 | ; | 32-bit |
| | PUSH | XAR7 | ; | 32-bit |
| | PUSH | XT | ; | 32-bit |
| | ; +8 = 16 cycles | | | |
| | | | | |
| | | | | |
| | | | | |
| | POP | XT | | |
| | POP | XAR7 | | |
| | POP | XAR6 | | |
| | POP | XAR5 | | |
| | POP | XAR4 | | |
| | POP | XAR3 | | |
| | POP | XAR2 | | |
| | POP | XAR1H;AR0H | | |
| | IRET | | | |
| | ; 16 cycles | | | |

3.5 Nonmaskable Interrupts

Nonmaskable interrupts cannot be blocked by any of the enable bits (the INTM bit, the DBGM bit, and enable bits in the IFR, IER, or DBGIER). The C28x immediately approves this type of interrupt and branches to the corresponding interrupt service routine. There is one exception to this rule: When the CPU is halted in stop mode (an emulation mode), no interrupts are serviced. Stop mode is described in section 7.4.1 on page 7-7.

The C28x nonmaskable interrupts include:

- Software interrupts (the INTR and TRAP instructions).
- Hardware interrupt NMI
- Illegal-instruction trap
- □ Hardware reset interrupt (RS)

The software interrupt instructions and $\overline{\text{NMI}}$ are described in this section. The illegal-instruction trap and reset are described in sections 3.6 and 3.7, respectively.

3.5.1 INTR Instruction

You can use the INTR instruction to initiate one of the following interrupts by name: $\overline{INT1}$ - $\overline{INT14}$, DLOGINT, RTOSINT and \overline{NMI} . For example, you can execute the interrupt service routine for $\overline{INT1}$ by using the following instruction: INTR INT1

Once an interrupt is initiated by the INTR instruction, how it is handled depends on which interrupt is specified:

- INT1-INT14, DLOGINT, and RTOSINT. These maskable interrupts have corresponding flag bits in the IFR. When a request for one of these interrupts is received at an external pin, the corresponding IFR bit is set and the interrupt must be enabled to be serviced. In contrast, when one of these interrupts is initiated by the INTR instruction, the IFR flag is not set, and the interrupt is serviced regardless of the value of any enable bits. However, in other respects, the INTR instruction and the hardware request are the same. For example, both clear the IFR bit that corresponds to the requested interrupt. For more details, see section 3.4 on page 3-11.
- NMI. Because this interrupt is nonmaskable, a hardware request at a pin and a software request with the INTR instruction lead to the same events. These events are identical to those that take place during a TRAP instruction (see section 3.5.2).

Chapter 6, *C28x Assembly Language Instructions*, contains a detailed description of the INTR instruction.

3.5.2 TRAP Instruction

You can use the TRAP instruction to initiate any interrupt, including one of the user-defined software interrupts (see USER1–USER12 in Table 3–1 on page 3-4). The TRAP instruction refers to one of the 32 interrupts by a number from 0 to 31. For example, you can execute the interrupt service routine for INT1 by using the following instruction:

TRAP #1

Regardless of whether the interrupt has bits set in the IFR and IER, neither the IFR nor the IER is affected by this instruction. Figure 3–5 shows a functional flow chart for an interrupt initiated by the TRAP instruction. For more details about the TRAP instruction, see Chapter 6, *C28x Assembly Language Instructions*.

Note:

The TRAP #0 instruction does not initiate a full reset. It only forces execution of the interrupt service routine that corresponds to the RESET interrupt vector.

Figure 3–5. Functional Flow Chart for an Interrupt Initiated by the TRAP Instruction



The following lists explains the steps shown in Figure 3-5:

- TRAP instruction fetched. The CPU fetches the TRAP instruction from program memory. The desired interrupt vector has been specified as an operand and is now encoded in the instruction word. At this stage, no other interrupts can be serviced until the CPU begins executing the interrupt service routine (step 9).
- 2) Empty the pipeline. The CPU completes any instructions that have reached or passed the decode 2 phase of the pipeline. Any instructions that have not reached this phase are flushed from the pipeline.
- 3) Increment and temporarily store PC. The PC is incremented by 1. This value is the *return address*, which is temporarily saved in an internal hold register. During the automatic context save (step 6), the return address is pushed onto the stack.
- 4) Fetch interrupt vector. The PC is set to point to the appropriate vector location (based on the VMAP bit and the interrupt), and the vector located at the PC address is loaded into the PC. (To determine which vector address has been assigned to each of the interrupts, see section 3.2, *Interrupt Vectors*, on page 3-4.)
- 5) Increment SP by 1. The stack pointer (SP) is incremented by 1 in preparation for the automatic context save (step 6). During the automatic context save, the CPU performs 32-bit accesses, which are aligned to even addresses. Incrementing SP by 1 ensures that the first 32-bit access will not overwrite the previous stack value.
- 6) Perform automatic context save. A number of register values are saved automatically to the stack. These registers are saved in pairs; each pair is saved in a single 32-bit operation. At the end of each 32-bit operation, the SP is incremented by 2. Table 3–3 shows the register pairs and the order in which they are saved. All 32-bit saves are even-word aligned. As shown in the table, the SP is not affected by this alignment.

| Save Register | | Bit 0 of Storage Address | | |
|------------------------|-------------------------------|---|--|--|
| Operation [†] | Pairs | SP Starts at Odd Address | SP Starts at Even Address | |
| | | $1 \leftarrow SP$ position before step 5 | 1 | |
| 1st | ST0 | 0 | $0 \leftarrow SP \text{ position before step 5}$ | |
| | т | 1 | 1 | |
| 2nd | AL | 0 | 0 | |
| | AH | 1 | 1 | |
| 3rd | PL [‡] | 0 | 0 | |
| | PH | 1 | 1 | |
| 4th | AR0 | 0 | 0 | |
| | AR1 | 1 | 1 | |
| 5th | ST1 | 0 | 0 | |
| | DP | 1 | 1 | |
| 6th | IER | 0 | 0 | |
| | DBGSTAT§ | 1 | 1 | |
| 7th | Return address (low half) | 0 | 0 | |
| | Return address (high half) | 1 | 1 | |
| | | $0 \leftarrow SP \text{ position after save}$ | 0 | |
| | | 1 | $1 \leftarrow SP$ position after save | |

Table 3–4. Register Pairs Saved and SP Positions for Context Saves

[†] All registers are saved as pairs, as shown.

[‡] The P register is saved with 0 shift (CPU ignores current state of the product shift mode bits, PM, in status register 0).

§ The DBGSTAT register contains special emulation information.

7) Set INTM and DBGM. Clear LOOP, EALLOW, and IDLESTAT. All these bits are in status register ST1 (described in section 2.4 on page 2-34). By setting INTM to 1, the CPU prevents maskable interrupts from disturbing the ISR. If you wish to nest interrupts, have the ISR clear the INTM bit. By setting DBGM to 1, the CPU prevents debug events from disturbing timecritical code in the ISR. If you do not want debug events blocked, have the ISR clear DBGM. The CPU clears LOOP, EALLOW, and IDLESTAT so that the ISR operates within a new context.

- 8) **Load PC with fetched vector.** The PC is loaded with the interrupt vector that was fetched in step 4. The vector forces program control to the ISR.
- 9) Execute interrupt service routine. The CPU executes the program code you have prepared to handle the interrupt. You may wish to have the interrupt service routine (ISR) save register values in addition to those saved in step 6. A typical ISR is shown in Example 3–1 on page 3-16.

If you want the ISR to inform external hardware that the interrupt is being serviced, you can use the IACK instruction to send an interrupt acknowledge signal. The IACK instruction accepts a 16-bit constant as an operand and drives this 16-bit value on the 16 least significant lines of the data-write bus, DWDB(15:0). For a detailed description of the IACK instruction, see Chapter 6, *C28x Assembly Language Instructions*.

10) **Program continues.** After the interrupt service routine is completed, the program continues where it left off (at the return address).

3.5.3 Hardware Interrupt NMI

An interrupt can be requested by way the $\overline{\text{NMI}}$ input pin, which must be driven low to initiate the interrupt. Although $\overline{\text{NMI}}$ cannot be masked, there are some debug execution states in which $\overline{\text{NMI}}$ is not serviced (see section 7.4, *Execution Control Modes*, on page 7-7). For more details on real-time mode, see section 7.4.2 on page 7-9. Once a valid request is detected on the $\overline{\text{NMI}}$ pin, the CPU handles the interrupt in the same manner as shown for the TRAP instruction (see section 3.5.2).

3.6 Illegal-Instruction Trap

Any one of the following three events causes an illegal-instruction trap:

- An invalid instruction is decoded (this includes invalid addressing modes).
- ☐ The opcode value 0000₁₆ is decoded. This opcode corresponds to the ITRAP0 instruction.
- ☐ The opcode value FFFF₁₆ is decoded. This opcode corresponds to the ITRAP1 instruction.

An illegal-instruction trap cannot be blocked, not even during emulation. Once initiated, an illegal-instruction trap operates the same as a TRAP #19 instruction. The handling of an interrupt initiated by the TRAP instruction is described in section 3.5.2. As part of its operation, the illegal-instruction trap saves the return address on the stack. Thus, you can detect the offending address by examining this saved value. For more information about the TRAP instruction, see Chapter 6, *C28x Assembly Language Instructions*.

3.7 Hardware Reset (RS)

When asserted, the reset input signal (\overline{RS}) places the CPU into a known state. As part of a hardware reset, all current operations are aborted, the pipeline is flushed, and the CPU registers are reset as shown in Table 3–5. Then the RESET interrupt vector is fetched and the corresponding interrupt service routine is executed. For the reset condition of signals, see the data sheet for your particular C28x DSP. Also see the your data sheet for specific information on the process for resetting your DSP. Although \overline{RS} cannot be masked, there are some debug execution states in which \overline{RS} is not serviced (see section 7.4, *Execution Control Modes*, on page 7-7).

Table 3–5. Registers After Reset

| Register | Bit(s) | Value After Reset | Comments |
|----------|---------|-------------------------|--|
| ACC | all | 0000 0000 ₁₆ | |
| XAR0 | all | 0000 0000 ₁₆ | |
| XAR1 | all | 0000 0000 ₁₆ | |
| XAR2 | all | 0000 0000 ₁₆ | |
| XAR3 | all | 0000 0000 ₁₆ | |
| XAR4 | all | 0000 0000 ₁₆ | |
| XAR5 | all | 0000 0000 ₁₆ | |
| XAR6 | all | 0000 0000 ₁₆ | |
| XAR7 | all | 0000 0000 ₁₆ | |
| DP | all | 0000 ₁₆ | DP points to data page 0. |
| IFR | 16 bits | 0000 ₁₆ | There are no pending interrupts. All interrupts pending at the time of reset have been cleared. |
| IER | 16 bits | 0000 ₁₆ | Maskable interrupts are disabled in the IER. |
| DBGIER | all | 0000 ₁₆ | Maskable interrupts are disabled in the DBGIER. |

| Register | Bit(s) | Value After Reset | Comments |
|----------|------------|-------------------------|--|
| Р | all | 0000 0000 ₁₆ | |
| PC | all | 3F FFC0 ₁₆ | PC is loaded with the reset interrupt vector at program-space address 00 0000 ₁₆ or 3F FFC0 ₁₆ . |
| RPC | all | 0000 ₁₆ | |
| SP | all | SP = 0x400 | SP points to address 0400. |
| ST0 | 0: SXM | 0 | Sign extension is suppressed. |
| | 1: OVM | 0 | Overflow mode is off. |
| | 2: TC | 0 | |
| | 3: C | 0 | |
| | 4: Z | 0 | |
| | 5: N | 0 | |
| | 6: V | 0 | |
| | 7–9: PM | 000 ₂ | The product shift mode is set to left-shift-by-1. |
| | 10–15: OVC | 00 0000 ₂ | |

Table 3–5. Registers After Reset (Continued)

| Register | Bit(s) | Value After Reset | Comments |
|----------|--------------|-------------------|---|
| ST1‡ | 0: INTM | 1 | Maskable interrupts are globally disabled. They cannot be serviced unless the C28x is in real-time mode with the CPU halted. |
| | 1: DBGM | 1 | Emulation accesses and events are disabled. |
| | 2: PAGE0 | 0 | PAGE0 stack addressing mode is enabled. PAGE0 direct addressing mode is disabled. |
| | 3: VMAP | 1 | The interrupt vectors are mapped to program- memory addresses 3F FFC0 ₁₆ -3F FFFF ₁₆ . |
| | 4: SPA | 0 | |
| | 5: LOOP | 0 | |
| | 6: EALLOW | 0 | Access to emulation regis- ters is disabled. |
| | 7: IDLESTAT | 0 | |
| | 8: AMODE | 0 | C27x/C28x addressing mode |
| | 9: OBJMODE | 0 | C27x object mode |
| | 10: Reserved | 0 | |
| | 11: M0M1MAP | 1 | |

Table 3–5. Registers After Reset (Continued)

| Register | Bit(s) | Value After Reset | Comments |
|----------|------------|-------------------------|--------------------------|
| | 12: XF | 0 | XFS output signal is low |
| | 13–15: ARP | 0002 | ARP points to AR0. |
| XT | all | 0000 0000 ₃₂ | |

Table 3–5. Registers After Reset (Continued)

Chapter 4

Pipeline

This chapter explains the operation of the C28x instruction pipeline. The pipeline contains hardware that prevents reads and writes at the same register or data-memory location from happening out of order. However, you can increase the efficiency of your programs if you take into account the operation of the pipeline. In addition, you should be aware of two types of pipeline conflicts the pipeline does not protect against and how you can avoid them (see section 4.5).

For more information about the instructions shown in examples throughout this chapter, see Chapter 6, *C28x Assembly Language Instructions*.

TopicPage4.1Pipelining of Instructions4-24.2Visualizing Pipeline Activity4-74.3Freezes in Pipeline Activity4-104.4Pipeline Protection4-12

Avoiding Unprotected Operations 4-16

4.5

4.1 Pipelining of Instructions

When executing a program, the C28x CPU performs these basic operations:

- Fetches instructions from program memory
- Decodes instructions
- Reads data values from memory or from CPU registers
- Executes instructions
- U Writes results to memory or to CPU registers

For efficiency, the C28x performs these operations in eight independent phases. Reads from memory are designed to be pipelined in two stages, which correspond to the two pipeline phases used by the CPU for each memory-read operation. At any time, there can be up to eight instructions being carried out, each in a different phase of completion. Following are descriptions of the eight phases in the order they occur. The address and data buses mentioned in these descriptions are introduced in section 1.4.1 on page 1-9.

- Fetch 1In the fetch 1 (F1) phase, the CPU drives a program-memory ad-(F1)dress on the 22-bit program address bus, PAB(21:0).
- Fetch 2 In the fetch 2 (F2) phase, the CPU reads from program memory
 (F2) by way of the program-read data bus, PRDB (31:0), and loads the instruction(s) into an instruction-fetch queue.
- Decode 1 The C28x supports both 32-bit and 16-bit instructions and an instruction can be aligned to an even or odd address. The decode 1 (D1) hardware identifies instruction boundaries in the instruction-fetch queue and determines the size of the next instruction to be executed. It also determines whether the instruction is a legal instruction.
| Decode 2 (D2) | The decode 2 (D2) hardware requests an instruction from the instruction-fetch queue. The requested instruction is loaded into the instruction register, where decoding is completed. Once an instruction reaches the D2 phase, it runs to completion. In this pipeline phase, the following tasks are performed: | | | | | | |
|------------------|--|---|--|--|--|--|--|
| | | If data is to be read from memory, the CPU generates the source address or addresses. | | | | | |
| | | If data is to be written to memory, the CPU generates the destination address. | | | | | |
| | | The address register arithmetic unit (ARAU) performs any required modifications to the stack pointer (SP) or to an auxiliary register and/or the auxiliary register pointer (ARP). | | | | | |
| | | If a program-flow discontinuity (such as a branch or an illegal-instruction trap) is required, it is taken. | | | | | |
| Read 1 (R1) | lf c driv | lata is to be read from memory, the read 1 (R1) hardware ves the address(es) on the appropriate address bus(es). | | | | | |
| Read 2 (R2) | lf d feto | ata was addressed in the R1 phase, the read 2 (R2) hardware ches that data by way of the appropriate data bus(es). | | | | | |
| Execute (E) | In t and Iog For ing arit of t shi the ten | In the execute (E) phase, the CPU performs all multiplier, shifter, and ALU operations. This includes all the prime arithmetic and logic operations involving the accumulator and product register. For operations that involve reading a value, modifying it, and writ- ing it back to the original location, the modification (typically an arithmetic or a logical operation) is performed during the E phase of the pipeline. Any CPU register values used by the multiplier, shifter, and ALU are read from the registers at the beginning of the E phase. A result that is to be written to a CPU register is writ- ten to the register at the end of the E phase. | | | | | |
| Write (W) | If a occ add The hai | transferred value or result is to be written to memory, the write curs in the write (W) phase. The CPU drives the destination dress, the appropriate write strobes, and the data to be written. e actual storing, which takes at least one more clock cycle, is ndled by memory wrappers or peripheral interface logic and is c visible as a part of the CPU pipeline. | | | | | |

Although every instruction passes through the eight phases, not every phase is active for a given instruction. Some instructions complete their operations in the decode 2 phase, others in the execute phase, and still others in the write phase. For example, instructions that do not read from memory perform no operations in the read phases, and instructions that do not write to memory perform no operation in the write phase.

Because different instructions perform modifications to memory and registers during different phases of their completion, an unprotected pipeline could lead to reads and writes at the same location happening out of the intended order. The CPU automatically adds inactive cycles to ensure that these reads and writes happen as intended. For more details about pipeline protection, see section 4.4 on page 4-12.

4.1.1 Decoupled Pipeline Segments

The fetch 1 through decode 1 (F1–D1) hardware acts independently of the decode 2 through write (D2–W) hardware. This allows the CPU to continue fetching instructions when the D2–W phases are halted. It also allows fetched instructions to continue through their D2–W phases when fetching of new instructions is delayed. Events that cause portions of the pipeline to halt are described in section 4.3.

Instructions in their fetch 1, fetch 2, and decode 1 phases are discarded if an interrupt or other program-flow discontinuity occurs. An instruction that reaches its decode 2 phase always runs to completion before any program-flow discontinuity is taken.

4.1.2 Instruction-Fetch Mechanism

Certain branch instructions perform prefetching. The first few instructions of the branch destination will be fetched but not allowed to reach DZ until it is known whether the discontinuity will be taken. The instruction-fetch mechanism is the hardware for the F1 and F2 pipeline phases. During the F1 phase, the mechanism drives an address on the program address bus (PAB). During the F2 phase, it reads from the program-read data bus (PRDB). While an instruction is read from program memory in the F2 phase, the address for the next fetch is placed on the program address bus (during the next F1 phase).

The instruction-fetch mechanism contains an instruction-fetch queue of four 32-bit registers. During the F2 phase, the fetched instruction is added to the queue, which behaves like a first-in, first-out (FIFO) buffer. The first instruction in the queue is the first to be executed. The instruction-fetch mechanism performs 32-bit fetches until the queue is full. When a program-flow discontinuity

(such as a branch or an interrupt) occurs, the queue is emptied. When the instruction at the bottom of the queue reaches its D2 phase, that instruction is passed to the instruction register for further decoding.

4.1.3 Address Counters FC, IC, and PC

Three program-address counters are involved in the fetching and execution of instructions:

- Fetch counter (FC). The fetch counter contains the address that is driven on the program address bus (PAB) in the F1 pipeline phase. The CPU continually increments the FC until the queue is full or the queue is emptied by a program-flow discontinuity. Generally, the FC holds an even address and is incremented by 2, to accommodate 32-bit fetches. The only exception to this is when the code after a discontinuity begins at an odd address. In this case, the FC holds the odd address. After performing a16-bit fetch at the odd address, the CPU increments the FC by 1 and resumes 32-bit fetching at even addresses.
- Instruction counter (IC). After the D1 hardware determines the instruction size (16-bit or 32-bit), it fills the instruction counter (IC) with the address of the next instruction to undergo D2 decoding. On an interrupt or call operation, the IC value represents the return address, which is saved to the stack, to auxiliary register XAR7, or to RPC.
- Program counter (PC). When a new address is loaded into the IC, the previous IC value is loaded into the PC. The program counter (PC) always contains the address of the instruction that has reached its D2 phase.

Example 4–1 shows the relationship between the pipeline and the address counters. Instruction 1 has reached its D2 phase (it has been passed to the instruction register). The PC points to the address from which instruction 1 was taken ($00\ 0050_{16}$). Instruction 2 has reached its D1 phase and will be executed next (assuming no program-flow discontinuity flushes the instruction-fetch queue). The IC points to the address from which instruction 2 was taken ($00\ 0051_{16}$). Instruction 3 is in its F2 phase. It has been transferred to the instruction-fetch queue but has not been decoded. Instructions 4 and 5 are each in their F1 phase. The FC address ($00\ 0054_{16}$) is being driven on the PAB. During the next 32-bit fetch, Instructions 4 and 5 will be transferred from addresses 00 0054_{16} and 00 0055_{16} to the queue.

| | Program memo | | | | |
|----------------------------|----------------|----------------|-------------------------------|--|--|
| | | | | | |
| IC → 00 0051 ₁₆ | Instruction 2 | Instruction 1 | 00 0050 ₁₆ ◀── I C | | |
| 00 0053 ₁₆ | Instru | uction 3 | 00 0052 ₁₆ | | |
| 00 0055 ₁₆ | Instruction 5 | Instruction 4 | 00 0054 ₁₆ | | |
| 00 0057 ₁₆ | Instruction 7 | Instruction 6 | 00 0056 ₁₆ | | |
| 00 0059 ₁₆ | Instruc | ction 8 | 00 0058 ₁₆ | | |
| 00 005B ₁₆ | Instruc | Instruction 9 | | | |
| 00 005D ₁₆ | Instruction 11 | Instruction 10 | 00 005C ₁₆ | | |
| | | | | | |
| | | • | | | |

| Example 4–1. I | Relationship Between | Pipeline and Address | Counters FC, IC, and PC |
|----------------|----------------------|----------------------|-------------------------|
|----------------|----------------------|----------------------|-------------------------|

Instruction-fetch queue (32 bits wide)



The remainder of this document refers almost exclusively to the PC. The FC and the IC are visible in only limited ways. For example, when a call is executed or an interrupt is initiated, the IC value is saved to the stack or to auxiliary register XAR7.

4.2 Visualizing Pipeline Activity

Consider Example 4–2, which lists eight instructions, I1–I8, and shows a diagram of the pipeline activity for those instructions. The F1 column shows addresses and the F2 column shows the instruction opcodes read at those addresses. During an instruction fetch, 32 bits are read, 16 bits from the specified address and 16 bits from the following address. The D1 column shows instructions being isolated in the instruction-fetch queue, and the D2 column indicates address generation and modification of address registers. The Instruction column shows the instructions that have reached the D2 phase. The R1 column shows addresses, and the R2 column shows the data values being read from those addresses. In the E column, the diagram shows results being written to the low half of the accumulator (AL). In the W column, address and a data values are driven simultaneously on the appropriate memory buses. For example, in the last active W phase of the diagram, the address 00 0205₁₆ is driven on the data-write address bus (DWAB), and the data value 1234₁₆ is driven on the data-write data bus (DWDB).

The highlighted blocks in Example 4–2 indicate the path taken by the instruction ADD AL,*AR0++. That path can be summarized as follows:

| Phase | Activity Shown |
|-------|---|
| F1 | Drive address 00 0042 ₁₆ on the program address bus (PAB). |
| F2 | Read the opcodes F347 and F348 from addresses 00 0042 ₁₆ and 00 0043 ₁₆ , respectively. |
| D1 | Isolate F348 in the instruction-fetch queue. |
| D2 | Use XAR0 = 0066_{16} to generate source address $0000\ 0066_{16}$ and then increment XAR0 to 0067_{16} . |
| R1 | Drive address 00 0066_{16} on the data-read data bus (DRDB). |
| R2 | Read the data value 1 from address 0000 0066 ₁₆ . |
| Е | Add 1 to content of AL (1230_{16}) and store result (1231_{16}) to AL. |
| W | No activity |

Example 4–2. Diagramming Pipeline Activity

| Address | Opcode | Inst | ructio | on | Initial Values |
|---------|--------|------|--------|---|-------------------------|
| 00 0040 | F345 | I1: | MOV | DP,#VarA ; DP = page that has VarA. | VarA address = 00 0203 |
| 00 0041 | F346 | I2: | MOV | AL,@VarA ; Move content of VarA to AL. | VarA = 1230 |
| 00 0042 | F347 | I3: | MOVB | AR0,#VarB ; AR0 points to VarB. | VarB address = 00 0066 |
| 00 0043 | F348 | I4: | ADD | AL,*XAR0++ ; Add content of VarB to | VarB = 0001 |
| | | | | ; AL, and add 1 to XAR0. | (VarB + 1) = 0003 |
| 00 0044 | F349 | I5: | MOV | <pre>@VarC,AL ; Replace content of VarC</pre> | (VarB + 2) = 0005 |
| | | | | ; with content of AL. | VarC address = 000204 |
| 00 0045 | F34A | I6: | ADD | AL,*XAR0++ ; Add content of (VarB + 1) | VarD address = 00 0205 |
| | | | | ; to AL, and add 1 to XAR0. | |
| 00 0046 | F34B | I7: | MOV | <pre>@VarD,AL ; Replace content of VarD</pre> | |
| | | | | ; with content of AL. | |
| 00 0047 | F34C | I8: | ADD | AL,*XAR0 ; Add content of (VarB + 2) | |
| | | | | ; to AL. | |

| F1 | F2 | D1 | Instruction | D2 | R1 | R2 | E | W |
|---------|-----------|------|-------------------------|--------------------------|---------|------|---------|---------|
| 00 0040 | | | | | | | | |
| | F346:F345 | | | | | | | |
| 00 0042 | | F345 | | | | | | |
| | F348:F347 | F346 | I1: MOV DP,#VarA | DP = 8 | | | | |
| 00 0044 | | F347 | I2: MOV AL,@VarA | Generate VarA address | - | | | |
| | F34A:F349 | F348 | I3: MOVB XAR0,#Var B | XAR0 = 66 | 00 0203 | - | | |
| 00 0046 | | F349 | I4: ADD AL,*XAR0+ + | XAR0 = 67 | - | 1230 | - | |
| | F34C:F34B | F34A | I5: MOV @VarC,AL | Generate VarC address | 00 0066 | - | AL=1230 | - |
| | | F34B | I6: ADD AL,*XAR0+ + | XAR0 = 68 | - | 0001 | - | - |
| | | F34C | I7: MOV @VarD,AL | Generate VarD address | 00 0067 | - | AL=1231 | - |
| | | | I8: ADD AL,*XAR0 | XAR0 = 68 | - | 0003 | - | - |
| | | | | | 00 0068 | - | AL=1234 | 00 0204 |
| | | | | | | | | 1231 |
| | | | | | | 0005 | - | - |
| | | | | | | | AL=1239 | 00 0205 |
| | | | | | | | | 1234 |

| F1 | F2 | D1 | Instruction | D2 | R1 | R2 | Е | W |
|----|----|----|-------------|----|----|----|---|---|
| | | | | | | | | - |

Note: The opcodes shown in the F2 and D1 columns were chosen for illustrative purposes; they are not the actual opcodes of the instructions shown.

The pipeline activity in Example 4–2 can also be represented by the simplified diagram in Example 4–3. This type of diagram is useful if your focus is on the path of each instruction rather than on specific pipeline events. In cycle 8, the pipeline is full: there is an instruction in every pipeline phase. Also, the effective execution time for each of these instructions is one cycle. Some instructions finish their activity at the D2 phase, some at the E phase, and some at the W phase. However, if you choose one phase as a reference, you can see that each instruction is in that phase for one cycle.

| F1 | F2 | D1 | D2 | R1 | R2 | E | W | Cycle |
|----|----|----|----|----|----|----|----|-------|
| I1 | | | | | | | | 1 |
| I2 | I1 | | | | | | | 2 |
| I3 | I2 | I1 | | | | | | 3 |
| I4 | I3 | I2 | I1 | | | | | 4 |
| I5 | I4 | I3 | I2 | Il | | | | 5 |
| I6 | I5 | I4 | I3 | I2 | I1 | | | 6 |
| I7 | I6 | I5 | I4 | I3 | I2 | I1 | | 7 |
| I8 | I7 | I6 | I5 | I4 | I3 | I2 | Il | 8 |
| | I8 | I7 | I6 | I5 | I4 | I3 | I2 | 9 |
| | | I8 | I7 | I6 | I5 | I4 | I3 | 10 |
| | | | I8 | I7 | I6 | I5 | I4 | 11 |
| | | | | I8 | I7 | IG | I5 | 12 |
| | | | | | I8 | I7 | I6 | 13 |
| | | | | | | I8 | I7 | 14 |
| | | | | | | | I8 | 15 |

Example 4–3. Simplified Diagram of Pipeline Activity

4.3 Freezes in Pipeline Activity

This section describes the two causes for freezes in pipeline activity:

- Wait states
- An instruction-not-available condition

4.3.1 Wait States

When the CPU requests a read from or write to a memory device or peripheral device, that device may take more time to finish the data transfer than the CPU allots by default. Each device must use one of the CPU *ready signals* to insert wait states into the data transfer when it needs more time. The CPU has three independent sets of ready signals: one set for reads from and writes to program space, a second set for reads from data space, and a third set for writes to data space. Wait-state requests freeze a portion of the pipeline if they are received during the F1, R1, or W phase of an instruction:

- Wait states in the F1 phase. The instruction-fetch mechanism halts until the wait states are completed. This halt effectively freezes activity for instructions in their F1, F2, and D1 phases. However, because the F1–D1 hardware and the D2–W hardware are decoupled, instructions that are in their D2–W phases continue to execute.
- Wait states in the R1 phase. All D2–W activities of the pipeline freeze. This is necessary because subsequent instructions can depend on the data-read taking place. Instruction fetching continues until the instructionfetch queue is full or a wait-state request is received during an F1 phase.
- Wait states in the W phase. All D2–W activity in the pipeline freezes. This is necessary because subsequent instructions may depend on the write operation happening first. Instruction fetching continues until the instruction-fetch queue is full or a wait-state request is received during an F1 phase.

4.3.2 Instruction-Not-Available Condition

The D2 hardware requests an instruction from the instruction-fetch queue. If a new instruction has been fetched and has completed its D1 phase, the instruction is loaded into the instruction register for more decoding. However, if a new instruction is *not* waiting in the queue, an instruction-not-available condition exists. Activity in the F1–D1 hardware continues. However, the activity in the D2–W hardware ceases until a new instruction is available. One time that an instruction-not-available condition will occur is when the first instruction after a discontinuity is at an odd address and has 32 bits. A *discontinuity* is a break in sequential program flow, generally caused by a branch, a call, a return, or an interrupt. When a discontinuity occurs, the instruction-fetch queue is emptied, and the CPU branches to a specified address. If the specified address is an odd address, a 16-bit fetch is performed at the odd address, followed by 32-bit fetches at subsequent even addresses. Thus, if the first instruction after a discontinuity is at an odd address and has 32 bits, two fetches are required to get the entire instruction. The D2–W hardware ceases until the instruction is ready to enter the D2 phase.

To avoid the delay where possible, you can begin each block of code with one or two (preferably two) 16-bit instructions:

FunctionA:

```
16-bit instruction ; First instruction
16-bit instruction ; Second instruction
32-bit instruction ; 32-bit instructions can start here
.
.
```

If you choose to use a 32-bit instruction as the first instruction of a function or subroutine, you can prevent a pipeline delay only by making sure the instruction begins at an even address.

Pipeline Protection 4.4

Instructions are being executed in parallel in the pipeline, and different instructions perform modifications to memory and registers during different phases of completion. In an unprotected pipeline, this could lead to pipeline conflictsreads and writes at the same location happening out of the intended order. However, the C28x pipeline has a mechanism that automatically protects against pipeline conflicts. There are two types of pipeline conflicts that can occur on the C28x:

Conflicts during reads and writes to the same data-space location

Register conflicts

The pipeline prevents these conflicts by adding inactive cycles between instructions that would cause the conflicts. Sections 4.4.1 and 4.4.2 explain the circumstances under which these pipeline-protection cycles are added and tells how to avoid them, so that you can reduce the number of inactive cycles in your programs.

4.4.1 Protection During Reads and Writes to the Same Data-Space Location

Consider two instructions, A and B. Instruction A writes a value to a memory location during its W phase. Instruction B must read that value from the same location during its R1 and R2 phases. Because the instructions are being executed in parallel, it is possible that the R1 phase of instruction B could occur before the W phase of instruction A. Without pipeline protection, instruction B could read too early and fetch the wrong value. The C28x pipeline prevents that read by holding instruction B in its D2 phase until instruction A is finished writing.

Example 4-4 shows a conflict between two instructions that are accessing the same data-memory location. The pipeline activity shown is for an unprotected pipeline. For convenience, the F1-D1 phases are not shown. I1 writes to VarA during cycle 5. Data memory completes the store in cycle 6. I2 should not read the data-memory location any sooner than cycle 7. However, I2 performs the read during cycle 4 (three cycles too early). To prevent this kind of conflict, the pipeline-protection mechanism would hold I2 in the D2 phase for 3 cycles. During these *pipeline-protection cycles*, no new operations occur.

| I1: MO I2: MO | V @VarA,AL V AH,@VarA | ; Write A ; Read sa | AL to data ame locatio | -memory lo on, store | cation value in AH |
|------------------|--------------------------|------------------------|---------------------------|-------------------------|-----------------------|
| DZ | KI | RZ | E | W | Cycle |
| I1 | | | | | 1 |
| I2 | I1 | | | | 2 |
| I2 | | I1 | | | 3 |
| I2 | | | I1 | | 4 |
| I2 | | | | I1 | 5 |
| | I2 | | | | 6 |
| | | I2 | | | 7 |
| | | | I2 | | 8 |

Example 4–4. Conflict Between a Read From and a Write to Same Memory Location

You can reduce or eliminate these types of pipeline-protection cycles if you can take other instructions in your program and insert them between the instructions that conflict. Of course, the inserted instructions must not cause conflicts of their own or cause improper execution of the instructions that follow them. For example, the code in Example 4-4 could be improved by moving a CLRC instruction to the position between the MOV instructions (assume that the instructions following CLRC SXM operate correctly with SXM = 0):

I1: MOV @VarA,AL ; Write AL to data-memory location CLRC SXM ; SXM = 0 (sign extension off) MOV AH,@VarA ; Read same location, store value in AH т2:

Inserting the CLRC instruction between I1 and I2 reduces the number of pipeline-protection cycles to two. Inserting two more instructions would remove the need for pipeline protection. As a general rule, if a read operation occurs within three instructions from a write operation to the same memory location, the pipeline protection mechanism adds at least one inactive cycle.

4.4.2 Protection Against Register Conflicts

All reads from and writes to CPU registers occur in either the D2 phase or the E phase of an instruction. A register conflict arises when an instruction attempts to read and/or modify the content of a register (in the D2 phase) before a previous instruction has written to that register (in the E phase).

The pipeline-protection mechanism resolves register conflicts by holding the later instruction in its D2 phase for as many cycles as needed (one to three). You do not have to consider register conflicts unless you wish to achieve maximum pipeline efficiency. If you choose to reduce the number of pipeline-protection cycles, you can identify the pipeline phases in which registers are accessed and try to move conflicting instructions away from each other.

Generally, a register conflict involves one of the address registers:

- 16-bit auxiliary registers AR0–AR7
- 32-bit auxiliary registers XAR0–XAR7
- 16-bit data page pointer (DP)
- 16-bit stack pointer (SP)

Example 4–5 shows a register conflict involving auxiliary register XAR0. The pipeline activity shown is for an *unprotected* pipeline, and for convenience, the F1–D1 phases are not shown. I1 writes to XAR0 at the end of cycle 4. I2 should not attempt to read XAR0 until cycle 5. However, I2 reads XAR0 (to generate an address) during cycle 2. To prevent this conflict, the pipeline-protection mechanism would hold I2 in the D2 phase for three cycles. During these cycles, no new operations occur.

Example 4–5. Register Conflict

| I1: MOVB AR0,@7 I2: MOV AH,*XAR0 | | ; Load AR ; the oper ; half of ; Load AH ; XAR0. | 20 with the and @7 and XAR0. I with the | e value ad d clear th value poi | ldressed by e upper nted to by |
|-------------------------------------|----|--|--|---------------------------------------|--------------------------------------|
| D2 | R1 | R2 | E | W | Cycle |
| I1 | | | | | 1 |
| I2 | I1 | | | | 2 |
| I2 | | I1 | | | 3 |
| | | | | | |

| I2 | | I1 | | | 3 |
|----|----|----|----|----|---|
| I2 | | | I1 | | 4 |
| I2 | | | | I1 | 5 |
| | I2 | | | | 6 |
| | | I2 | | | 7 |
| | | | | | |

You can reduce or eliminate pipeline-protection cycles due to a register conflict by inserting other instructions between the instructions that cause the conflict. For example, the code in Example 4–5 could be improved by moving two other instructions from elsewhere in the program (assume that the instructions following SETC SXM operate correctly with PM = 1 and SXM = 1):

| I1: | MOVB AR0,@7 | ; Load AR0 with the value addressed by |
|-----|--------------|--|
| | | ; the operand @7 and clear the upper |
| | | ; half of XAR0. |
| | SPM 0 | ; PM = 1 (no product shift) |
| | SETC SXM | ; SXM = 1 (sign extension on) |
| I2: | MOV AH,*XAR0 | ; Load AH with the value pointed to by |
| | | ; AR0. |

Inserting the SPM and SETC instructions reduces the number of pipelineprotection cycles to one. Inserting one more instruction would remove the need for pipeline protection. As a general rule, if a read operation occurs within three instructions from a write operation to the same register, the pipeline-protection mechanism adds at least one inactive cycle.

4.5 Avoiding Unprotected Operations

This section describes pipeline conflicts that the pipeline-protection mechanism does not protect against. These conflicts are avoidable, and this section offers suggestions for avoiding them.

4.5.1 Unprotected Program-Space Reads and Writes

The pipeline protects only register and data-space reads and writes. It does not protect the program-space reads done by the PREAD and MAC instructions or the program-space write done by the PWRITE instruction. Be careful with these instructions when using them to access a memory block that is shared by data space and program space.

As an example, suppose a memory location can be accessed at address $00\ 0D50_{16}$ in program space and address $0000\ 0D50_{16}$ in data space. Consider the following lines of code:

```
; XAR7 = 000D50 in program space
; Data1 = 000D50 in data space
ADD @Data1,AH ; Store AH to data-memory location
                ; Data1.
PREAD @AR1,*XAR7 ; Load AR1 from program-memory
               ; location given by XAR7.
```

The operands @Data1 and *XAR7 are referencing the same location, but the pipeline cannot interpret this fact. The PREAD instruction reads from the memory location (in the R2 phase) before the ADD writes to the memory location (in the W phase).

However, the PREAD is not necessary in this program. Because the location can be accessed by an instruction that reads from data space, you can use another instruction, such as a MOV instruction:

ADD @Data1,AH ; Store AH to memory location Data1. MOV AR1,*XAR7 ; Load AR1 from memory location ; given by XAR7.

4.5.2 An Access to One Location That Affects Another Location

If an access to one location affects another location, you may need to correct your program to prevent a pipeline conflict. You only need to be concerned about this kind of pipeline conflict if you are addressing a location outside of a protected address range. (See section 4.5.3.). Consider the following example:

This program causes a misread. The TBIT instruction reads bit 15 (in the R2 phase) before the MOV instruction writes to bit 15 (in the W phase). If the TBIT instruction reads a 1, the code prematurely ends the loop. Because DataA and DataB reference different data-memory locations, the pipeline does not identify this conflict.

However, you can correct this type of error by inserting two or more NOP (no operation) instructions to allow for the delay between the write to DataA and the change to bit 15 of DataB. For example, if a 2-cycle delay is sufficient, you can fix the previous code as follows:

4.5.3 Write Followed By Read Protection Mode

The CPU contains a write followed by read protection mode to ensure that any read operation that follows a write operation within a protected address range is executed as written by delaying the read operation until the write is initiated.

See your device data sheet for device-specific information about which memory region is write-followed-by-read protected.

The PROTSTART(15:0) and PROTRANGE(15:0) input signals set the protection range. The PROTRANGE(15:0) value is a binary multiple with the smallest block size being 64 words, and the largest being 4M words (64 words, 128 words, 256 words ...1M words, 2M words, 4M words). The PROTSTART address must always be a multiple of the chosen range. For example, if a 4K block size is selected, then the start address must be a multiple of 4K.

The ENPROT signal enables this feature (when set high), it disables this feature (when set low)

All of the above signals are latched on every cycle. The above signals are connected to registers and can be changed within the application program.

The above mechanism only works for reads that follow writes to the protected area. Reads and write sequences to unprotected areas are not affected, as shown in the following examples.

| Example 1: | <pre>write protected_area write protected_area write protected_area</pre> | | |
|------------|---|-------------|-------------------------------|
| | | <- | pipe protection (3 cycles) |
| | read protected_area | | |
| Example 2: | <pre>write protected_area write protected_area write protected_area</pre> | | |
| | | <- | no pipe protection invoked |
| | read non_protected_area | | |
| | | <- | pipe protection (2 cycles) |
| | read protected_area read protected_area | | |
| Example 3: | <pre>write non_protected_area write non_protected_area write non_protected_area</pre> | 1 1 1 | |
| | | <- | no pipe protection invoked |
| | read protected_area | | |

Chapter 5

C28x Addressing Modes

This chapter describes the addressing modes of the C28x and provides examples.

Topic

Page

| 5.1 | Types of Addressing Modes 5-2 |
|------|--|
| 5.2 | Addressing Modes Select Bit (AMODE) 5-4 |
| 5.3 | Assembler/Compiler Tracking of AMODE Bit 5-7 |
| 5.4 | Direct Addressing Modes (DP) 5-8 |
| 5.5 | Stack Addressing Modes (SP) 5-9 |
| 5.6 | Indirect Addressing Modes 5-10 |
| 5.7 | Register Addressing Modes 5-25 |
| 5.8 | Data/Program/IO space Immediate Addressing Modes |
| 5.9 | Program Space Indirect Addressing Modes 5-30 |
| 5.10 | Byte Addressing Modes 5-31 |
| 5.11 | Alignment of 32-Bit Operations |

5.1 Types of Addressing Modes

The C28x CPU supports four basic types of addressing modes:

Direct Addressing Mode

DP (data page pointer): In this mode, the 16-bit DP register behaves like a fixed page pointer. The instruction supplies a 6-bit or 7-bit offset field, which is concatenated with the value in the DP register. This type of addressing is useful for accessing fixed address data structures, such as peripheral registers and global or static variables in C/C++.

Stack Addressing Mode

SP (stack pointer): In this mode, the 16-bit SP pointer is used to access information on the software stack. The software stack grows from low to high memory on the C28x and the stack pointer always points to the next empty location. The instruction supplies a 6-bit offset field that is sub-tracted from the current stack pointer value for accessing data on the stack or the stack pointer can be post-incremented or pre-decremented when pushing and popping data from the stack, respectively.

Indirect Addressing Mode

XAR0 to XAR7 (auxiliary register pointers): In this mode, the 32-bit XARn registers behave as generic data pointers. The instruction can direct to post-increment, pre/post-decrement, or index from the current register contents with either a 3-bit immediate offset field or with the contents of another 16-bit register.

Register Addressing Mode

In this mode, another register can be the source or destination operand of an access. This enables register-to-register operations in the C28x architecture.

On most C28x instructions, an 8-bit field in the instruction op-code selects the addressing mode to use and what modification to make to that mode. In the C28x instruction set, this field is referred to as:

loc16

Selects Direct/Stack/Indirect/Register addressing mode for 16-bit data access.

loc32

Selects Direct/Stack/Indirect/Register addressing mode for 32-bit data access.

An example C28x instruction description, which uses the above, would be:

ADD AL,loc16

Take the 16-bit contents of AL register, add the contents of 16-bit location specified by the "loc16" field and store the contents in AL register.

□ ADDL loc32,ACC

Take the 32-bit contents of the location pointed to by the "loc32" field, add the contents of the 32-bit ACC register, and store the result back into the location specified by the "loc32" field.

Other types of addressing modes supported are:

Data/Program/IO Space Immediate Addressing Modes:

In this mode, the address of the memory operand is embedded in the instruction.

Program Space Indirect Addressing Modes:

Some instructions can access a memory operand located in program space using an indirect pointer. Since memory is unified on the C28x CPU, this enables the reading of two operands in a single cycle.

Only a small number of instructions use the above modes and typically they are in combination with the "loc16/loc32" modes.

The following sections contain detailed descriptions of the addressing modes with example instructions. For more information about the instructions shown in examples throughout this chapter, see *Chapter 6, Assembly Language Instructions*.

5.2 Addressing Modes Select Bit (AMODE)

To accommodate various types of addressing modes, an addressing mode bit (AMODE) selects the decoding of the 8-bit field (loc16/loc32). This bit is found in Status Register 1 (ST1). The addressing modes have been broadly classified as follows:

AMODE = 0

This is the default mode on reset and is the mode used by the C28x C/C++ compiler. This mode is not fully compatible to the C2xLP CPU addressing modes. The data page pointer offset is 6-bits (it is 7-bits on the C2xLP) and not all of the indirect addressing modes are supported.

AMODE = 1

This mode contains addressing modes that are fully compatible to the C2xLP device. The data page pointer offset is increased to 7-bits and all of the indirect addressing modes available on the C2xLP are supported.

The available addressing modes, for the "loc16" or "loc32" field, are summarized in Table 5–1.

Table 5–1. Addressing Modes for "loc16" or "loc32"

| AMODE = 0 | | AMODE = 1 | |
|----------------------|------------------------|--------------|----------------------|
| 8-Bit Decode | "loc16/loc32" Syntax | 8-Bit Decode | "loc16/loc32" Syntax |
| Direct Addressing Mc | odes (DP): | | |
| 0 0 III III | @6bit | O I III III | @@7bit |
| Stack Addressing Mod | les (SP): | | |
| 0 1 III III | *-SP[6bit] | | |
| 1 0 111 101 | *SP++ | 1 0 111 101 | *SP++ |
| 1 0 111 110 | *SP | 1 0 111 110 | *SP |
| C28x Indirect Addres | ssing Modes (XAR0 to 3 | XAR7): | |
| 1 0 000 AAA | *XARn++ | 1 0 000 AAA | *XARn++ |
| 1 0 001 AAA | *XARn | 1 0 001 AAA | *XARn |
| 1 0 010 AAA | *+XARn[AR0] | 1 0 010 AAA | *+XARn[AR0] |
| 1 0 011 AAA | *+XARn[AR1] | 1 0 011 AAA | *+XARn[AR1] |
| 1 1 III AAA | *+XARn[3bit] | | |

| AMOE | DE = 0 | AMODE = 1 | | |
|--|-----------------------|--------------|----------------------|--|
| 8-Bit Decode | "loc16/loc32" Syntax | 8-Bit Decode | "loc16/loc32" Syntax | |
| C2xLP Indirect Addre | ssing Modes (ARP, XA) | R0 to XAR7): | | |
| 1 0 111 000 | * | 1 0 111 000 | * | |
| 1 0 111 001 | *++ | 1 0 111 001 | *++ | |
| 1 0 111 010 | * | 1 0 111 010 | * | |
| 1 0 111 011 | *0++ | 1 0 111 011 | *0++ | |
| 1 0 111 100 | *0 | 1 0 111 100 | *0 | |
| 1 0 101 110 | *BR0++ | 1 0 101 110 | *BR0++ | |
| 1 0 101 111 | *BR0 | 1 0 101 111 | *BR0 | |
| 1 0 110 RRR | *,ARPn | 1 0 110 RRR | *,ARPn | |
| | | 1 1 000 RRR | *++, ARPn | |
| | | 1 1 001 RRR | *, ARPn | |
| | | 1 1 010 RRR | *0++,ARPn | |
| | | 1 1 011 RRR | *0,ARPn | |
| | | 1 1 100 RRR | *BR0++,ARPn | |
| | | 1 1 101 RRR | *BR0,ARPn | |
| Circular Indirect Ad | dressing Modes (XAR6) | , XAR1): | | |
| 1 0 111 111 | *AR6%++ | 1 0 111 111 | *+XAR6[AR1%++] | |
| 32-Bit Register Addressing Modes (XAR0 to XAR7, ACC, P, XT): | | | | |
| 1 0 100 AAA | @XARn | 1 0 100 AAA | @XARn | |
| 1 0 101 001 | @ACC | 1 0 101 001 | @ACC | |
| 1 0 101 011 | @P | 1 0 101 011 | @P | |
| 1 0 101 100 | @XT | 1 0 101 100 | @XT | |

Table 5–1. Addressing Modes for "loc16" or "loc32"

| AMO | DE = 0 | AMODE = 1 | | |
|----------------------|----------------------|----------------------|----------------------|--|
| 8-Bit Decode | "loc16/loc32" Syntax | 8-Bit Decode | "loc16/loc32" Syntax | |
| 16-Bit Register Addr | essing Modes (AR0 to | AR7, AH, AL, PH, PL, | TH, SP): | |
| 1 0 100 AAA | @ARn | 1 0 100 AAA | @ARn | |
| 1 0 101 000 | @AH | 1 0 101 000 | @AH | |
| 1 0 101 001 | @AL | 1 0 101 001 | @AL | |
| 1 0 101 010 | @PH | 1 0 101 010 | @PH | |
| 1 0 101 011 | @PL | 1 0 101 011 | @PL | |
| 1 0 101 100 | @TH | 1 0 101 100 | @TH | |
| 1 0 101 101 | @SP | 1 0 101 101 | @SP | |

Table 5–1. Addressing Modes for "loc16" or "loc32"

In the "C28x Indirect" addressing modes, the auxiliary register pointer used in the addressing mode is implicitly specified. In the "C2xLP Indirect" addressing modes, a 3-bit pointer called the auxiliary register pointer (ARP) is used to select which of the auxiliary registers is currently used and which pointer is used in the next operation.

The examples below illustrate the differences between the "C28x Indirect" and "C2xLP Indirect" addressing modes:

ADD AL,*XAR4++

Read the contents of 16-bit memory location pointed to by register XAR4, add the contents to AL register. Post-increment the contents of XAR4 by 1.

□ ADD AL,*++

Assume ARP pointer in ST1 contains the value 4. Read the contents of 16-bit memory location pointed to by register XAR4, add the contents to AL register. Post-increment the contents of XAR4 by 1.

□ ADD AL,*++,ARP5

Assume ARP pointer in ST1 contains the value 4. Read the contents of 16-bit memory location pointed to by register XAR4, add the contents to AL register. Post-increment the contents of XAR4 by 1. Set the ARP pointer to 5. Now it points to XAR5.

On the C28x instruction syntax, the destination operand is always on the left and the source operands are always on the right.

5.3 Assembler/Compiler Tracking of AMODE Bit

The compiler will always assume the addressing mode is set to AMODE = 0and therefore will only use addressing modes that are valid for AMODE = 0. The assembler can be instructed, via the command line options, to default to either AMODE = 0 or AMODE = 1. The command line options are:

| -v28 | Assumes AMODE = 0 (C28x addressing modes). |
|-----------|--|
| -v28 -m20 | Assumes AMODE = 1 (full C2xLP compatible addressing modes. |

Additionally, the assembler allows directives to be embedded within a file to instruct the assembler to override the default mode and change syntax checking to the new address mode setting:

| .c28_amode | Tells assembler that any code that follows assumes AMODE = 0 (C28x addressing modes). |
|------------|---|
| .lp_amode | Tells assembler that any code that follows assumes AMODE = 1 (full C2xLP compatible addressing modes) |

The above directives cannot be nested. The above directives can be used as follows within an assembly program:

| ; File assemble | d | using "-v28" option (assume AMODE = 0): |
|-----------------|---|---|
| | ; | This section of code can only use AMODE = 0 |
| | ; | addressing modes |
| | | |
| | | |
| • | | |
| • | | |
| SETC AMODE | ; | Change to AMODE = 1 |
| .lp_amode | ; | Tell assembler to check for AMODE = 1 syntax |
| | ; | This section of code can only use AMODE = 1 |
| | ; | addressing modes |
| • | | |
| • | | |
| • | | |
| • | | |
| CLRC AMODE | ; | Revert back to $AMODE = 0$ |
| .c28_amode | ; | Tell assembler to check for AMODE = 1 syntax |
| | ; | This section of code can only use $AMODE = 0$ |
| | ; | addressing modes |
| • | | |
| • | | |
| • | | |
| • | | |
| ; End of file. | | |

C28x Addressing Modes 5-7

5.4 Direct Addressing Modes (DP)

| AMODE | "loc16/loc32" Syntax | Description | |
|---------------------|----------------------|--|--|
| 0 | @6bit | 32bitDataAddr(31:22) = 0 | |
| | | 32bitDataAddr(21:6) = DP(15:0) | |
| | | 32bitDataAddr(5:0) = 6bit | |
| | | Note: The 6-bit offset value is concatenated with the 16-bit DP register. The offset value enables 0 to 63 words to be addressed relative to the current DP register value. | |
| Example(s): MOVW | DP,#VarA ; Lo | ad DP pointer with page value containing VarA | |

| MOVW | DP,#VarA | ; Load DP pointer with page value containing VarA | 7 |
|------|----------|---|-----|
| ADD | AL,@VarA | ; Add memory location VarA to register AL | |
| MOV | @VarB,AL | ; Store AL into memory location VarB | |
| | | ; VarB is located in the same 64-word page as Var | ſΑ |
| MOVW | DP,#VarC | ; Load DP pointer with page value containing VarC | 7 1 |
| SUB | AL,@VarC | ; Subtract memory location VarC from register AL | |
| MOV | @VarD,AL | ; Store AL into memory location VarD | |
| | | ; VarC is located in the same 64-word page as Var | сD |
| | | ; VarC & D are in different pages than VarA & B | |

| AMODE | "loc16/loc32" Syntax | Description | |
|-------|----------------------|---|--|
| 1 | @@7bit | 32bitDataAddr(31:22) = 0 | |
| | | 32bitDataAddr(21:7) = DP(15:1) | |
| | | 32bitDataAddr(6:0) = 7bit | |
| | | Note: The 7-bit offset value is concatenated with the upper 15-bits of the DP register. Bit 0 of DP register is ignored and is not affected by the operation. The offset value enables 0 to 127 words to be addressed relative to the current DP register value. | |

| ; Make sure AMODE = 1 |
|---|
| ; Tell assembler that AMODE = 1 |
| ; Load DP pointer with page value containing VarA |
| ; Add memory location VarA to register AL |
| ; Store AL into memory location VarB |
| ; VarB is located in the same 128-word page as VarA |
| ; Load DP pointer with page value containing VarC |
| ; Subtract memory location VarC from register AL |
| ; Store AL into memory location VarD |
| ; VarC is located in the same 128-word page as VarD |
| ; VarC & D are in different pages than VarA & B |
| |

Note: The direct addressing mode can access only the lower 4M of data address space on the C28x device.

5.5 Stack Addressing Modes (SP)

| AMODE | "loc16/loc32" Syntax | Descr | iption | |
|-------|----------------------|---------------------------------|---|--|
| 0 | *-SP[6bit] | 32bitDataAddr(31:16) = 0x0000 | | |
| | | 32bitDataAddr(15:0) = SP – 6bit | | |
| | | Note: | The 6-bit offset value is subtracted from the current 16-bit SP register val- ue. The offset value enables 0 to 63 words to be addressed relative to the current SP register value. | |

Example(s):

| _ | | | | |
|---|------|--------------|---|--|
| | ADD | AL,*-SP[5] | ; | Add 16-bit contents from stack location |
| | | | ; | -5 words from top of stack to AL register |
| | MOV | *-SP[8],AL | ; | Store 16-bit AL register to stack location |
| | | | ; | -8 words from top of stack |
| | ADDL | ACC,*-SP[12] | ; | Add 32-bit contents from stack location |
| | | | ; | -12 words from top of stack to ACC register. |
| | MOVL | *-SP[34],ACC | ; | Store 32-bit ACC register to stack location |
| | | | ; | -34 words from top of stack |
| | | | | |

| AMODE | "loc16/loc32" Syntax | Description |
|-------|----------------------|-------------------------------|
| Х | *SP++ | 32bitDataAddr(31:16) = 0x0000 |
| | | 32bitDataAddr(15:0) = SP |
| | | if(loc16), $SP = SP + 1$ |
| | | if(loc32), SP = SP + 2 |

Example(s): MOV *

MOVL

| SP++,AL | ; | Push contents | of | 16-bit | AL | register | onto | b top |
|---------|---|---------------|----|--------|----|----------|------|-------|
| | ; | of stack | | | | | | |
| *SP++,P | ; | Push contents | of | 32-bit | Ρ | register | onto | top |
| | ; | of stack | | | | | | |

| AMODE | "loc16/loc32" Syntax | Description |
|-------|----------------------|-------------------------------|
| Х | *SP | if(loc16), SP = SP – 1 |
| | | if(loc32), $SP = SP - 2$ |
| | | 32bitDataAddr(31:16) = 0x0000 |
| | | 32bitDataAddr(15:0) = SP |

Example(s):

| ADD AL,*SP | ; Pop contents from top of stack and add to 16-b | it |
|---------------|--|----|
| | ; AL register | |
| MOVL ACC, *SP | ; Pop contents from top of stack and store in | |
| | ; 32-bit ACC register | |



5.6 Indirect Addressing Modes

This section includes indirect addressing modes for the 28x and 2xLP devices. It also includes circular indirect addressing modes.

| 7) |
|----|
| |

| AMO | DE | "loc16/loc32" Syntax | Description | | | |
|-------|---------------------|---|--|--|--|--|
| Х | | *XARn++ | ARP = n | | | |
| | | | 32bitDataAddr(31:0) = XARn | | | |
| | | | if(loc16), XARn = XARn + 1 | | | |
| | | | if(loc32), XARn = XARn + 2 | | | |
| Examp | Example(s): | | | | | |
| | MOVL MOVL MOV | XAR2,#Array1 XAR3,#Array2 @AR0,#N-1 | ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N | | | |
| Loop: | | | | | | |
| | MOVL | ACC,*XAR2++ | ; Load ACC with location pointed to by XAR2, ; post-increment XAR2 | | | |
| | MOVL | *XAR3++, ACC | ; Store ACC into location pointed to by XAR3, ; post-increment XAR3 | | | |
| | BANZ | Loop,AR0 | ; Loop until AR0 == 0, post-decrement AR0 | | | |

| AMODE | "loc16/loc32" Syntax | Description |
|-------|----------------------|----------------------------|
| Х | *XARn | ARP = n |
| | | if(loc16), XARn = XARn – 1 |
| | | if(loc32), XARn = XARn – 2 |
| | | 32bitDataAddr(31:0) = XARn |

Example(s):

| - | • • | | | |
|-------|------|------------------|---|---|
| | MOVL | XAR2,#Array1+N*2 | ; | Load XAR2 with end address of Array1 |
| | MOVL | XAR3,#Array2+N*2 | ; | Load XAR3 with end address of Array2 |
| | MOV | @AR0,#N-1 | ; | Load AR0 with loop count N |
| Loop: | | | | |
| _ | MOVL | ACC, *XAR2 | ; | Pre-decrement XAR2, |
| | | | ; | load ACC with location pointed to by XAR2 |
| | MOVL | *XAR3,ACC | ; | Pre-decrement XAR3, |
| | | | ; | store ACC into location pointed to by XAR3, |
| | BANZ | Loop,AR0 | ; | Loop until AR0 == 0, post-decrement AR0 |
| | | | | |

| AMODE | "loc16/loc32" Syntax | Descr | Description | | |
|-------|----------------------|----------------------------------|--|--|--|
| Х | *+XARn[AR0] | ARP = n | | | |
| | | 32bitDataAddr(31:0) = XARn + AR0 | | | |
| | | Note: | The lower 16-bits of XAR0 are added to the selected 32-bit register. Upper 16-bits of XAR0 are ignored. AR0 is treated as an unsigned 16-bit value. Overflow into the upper 16-bits of XARn can occur. | | |

Example(s):

| <u> </u> | - () | | |
|----------|-------|------------------|---|
| | MOVW | DP,#Array1Ptr | ; Point to Array1 Pointer location |
| | MOVL | XAR2,@Array1Ptr | ; Load XAR2 with pointer to Array1 |
| | MOVB | XAR0,#16 | ; $AR0 = 16$, $AR0H = 0$ |
| | MOVB | XAR1,#68 | ; AR1 = 68, AR1H = 0 |
| | MOVL | ACC, *+XAR2[AR0] | ;; Swap contents of location Array1[16] |
| | MOVL | P,*+XAR2[AR1] | ;; with the contents of location Array1[68] |
| | MOVL | *+XAR2[AR1],ACC | ;; |
| | MOVL | *+XAR2[AR0], P | ;; |

| AMODE | "loc16/loc32" Syntax | Descr | Description | |
|-------|----------------------|---------|--|--|
| Х | *+XARn[AR1] | ARP = n | | |
| | | 32bitD | ataAddr(31:0) = XARn + AR1 | |
| | | Note: | The lower 16-bits of XAR0 are added to the selected 32-bit register. Upper 16-bits of XAR0 are ignored. AR0 is treated as an unsigned 16-bit value. Overflow into the upper 16-bits of XARn can occur. | |

Example(s):

| MOVW | DP,#Array1Ptr | ; Point to Array1 Pointer location |
|------|------------------|---|
| MOVL | XAR2,@Array1Ptr | ; Load XAR2 with pointer to Array1 |
| MOVB | XAR0,#16 | ; AR0 = 16, AR0H = 0 |
| MOVB | XAR1,#68 | ; $AR1 = 68$, $AR1H = 0$ |
| MOVL | ACC, *+XAR2[AR0] | ;; Swap contents of location Array1[16] |
| MOVL | P,*+XAR2[AR1] | ;; with the contents of location Array1[68] |
| MOVL | *+XAR2[AR1],ACC | ;; |
| MOVL | *+XAR2[AR0],P | ;; |

| AMODE | "loc16/loc32" Syntax | Description | |
|-------|----------------------|--|--|
| Х | *+XARn[3bit] | ARP = n | |
| | | 32bitDataAddr(31:0) = XARn + 3bit | |
| | | Note: The immediate value is treated as an unsigned 3-bit value. | |

Example(s):

MOVW DP,#Array1Ptr ; Point to Array1 Pointer location MOVL XAR2,@Array1Ptr ; Load XAR2 with pointer to Array1 MOVL ACC,*+XAR2[2] ;; Swap contents of location Array1[2] MOVL P,*+XAR2[5] ;; with the contents of location Array1[5] MOVL *+XAR2[5],ACC ;; MOVL *+XAR2[2],P ;;

Note: The assembler also accepts "*XARn" as an addressing mode. This is the same encoding as the "*+XARn[0]" mode.

| AMODE | "loc16/loc32" Syntax | Description | |
|-----------|----------------------|---|--|
| Х | * | 32bitDataAddr(31:0) = XAR(ARP) | |
| | | Note: The XARn register used is the register pointed to by the current value in the ARP pointer. ARP = 0, points to XAR0, ARP = 1, points to XAR1 and so on. | |
| Example(s |): | | |
| MOVZ | DP,#RegAPtr | ; Load DP with page address containing RegAPtr | |
| MOVZ | AR2,@RegAPtr | ; Load AR2 with contents of RegAPtr, AR2H = 0 | |
| MOVZ | AR3,@RegBPtr | ; Load AR3 with contents of RegBPtr, AR3H = 0 | |
| | | ; RegAPtr and RegBPtr are located in the same | |
| | | ; 128 word data page. Both are located in | |
| | | ; the low 64K of data memory space. | |
| NOP | *,ARP2 | ; Set ARP pointer to point to XAR2 | |
| MOV | *,#0x0404 | ; Store 0x0404 into location pointed by XAR2 | |
| NOP | *,ARP3 | ; Set ARP pointer to point to XAR3 | |
| MOV | *,#0x8000 | ; Store 0x8000 into location pointed by XAR3 | |

5.6.2 C2xLP Indirect Addressing Modes (ARP, XAR0 to XAR7)

| AMODE | "loc16/loc32" Syntax | Description | |
|-----------|----------------------|--|--|
| Х | *, ARPn | 32bitDataAddr(31:0) = XAR(ARP) | |
| | | ARP = n | |
| Example(s |): | | |
| MOVZ | DP,#RegAPtr | ; Load DP with page address containing RegAPtr | |
| MOVZ | AR2,@RegAPtr | ; Load AR2 with contents of RegAPtr, AR2H = 0 | |
| MOVZ | AR3,@RegBPtr | ; Load AR3 with contents of RegBPtr, AR3H = 0 | |
| | | ; RegAPtr and RegBPtr are located in the same | |
| | | ; 128 word data page. Both are located in | |
| | | ; the low 64K of data memory space. | |
| NOP | *,ARP2 | ; Set ARP pointer to point to XAR2 | |
| MOV | *,#0x0404,ARP3 | ; Store 0x0404 into location pointed by XAR2, | |
| | | ; Set ARP pointer to point to XAR3 | |
| MOV | *,#0x8000 | ; Store 0x8000 into location pointed by XAR3 | |

| | 1 | | | | |
|---|--|--|--|--|--|
| AMODE "loc16/loc32" Syntax | | "loc16/loc32" Syntax | Description | | |
| Х | | *++ | 32bitDataAddr(31:0) = XAR(ARP) | | |
| | | | if(loc16), XAR(ARP) = XAR(ARP) + 1 | | |
| | | | if(loc32), XAR(ARP) = XAR(ARP) + 2 | | |
| Example | e(s): | | | | |
| ľ | MOVL | XAR2,#Array1 | ; Load XAR2 with start address of Array1 | | |
| M | MOVL | XAR3,#Array2 | ; Load XAR3 with start address of Array2 | | |
| M | MOV | @AR0,#N-1 | ; Load AR0 with loop count N | | |
| Loop: | | | | | |
| N | NOP | *,ARP2 | ; Set ARP pointer to point to XAR2 | | |
| Μ | MOVL | ACC, *++ | ; Load ACC with location pointed to by XAR2, | | |
| N | NOP | *.ARP3 | : Set ARP pointer to point to XAR3 | | |
| M | MOVL | *++, ACC | ; Store ACC into location pointed to by XAR3, | | |
| | | | ; post-increment XAR3 | | |
| N | NOP | *,ARPO | ; Set ARP pointer to point to XAR0 | | |
| Х | XBANZ | Loop,* | ; Loop until AR0 == 0, post-decrement AR0 | | |
| | - 1 | " | Description | | |
| ANIOD | ין שי | 10C16/10C32" Syntax | | | |
| X | | *++, ARPn | 32bitDataAddr(31:0) = XAR(ARP) | | |
| X | | *++ , ARPn | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 | | |
| X | | *++, ARPn | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 | | |
| X | e(s): | *++, ARPn | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 | | |
| Example M | e(s): | *++, ARPn XAR2, #Arrav1 | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 : Load XAR2 with start address of Arrav1 | | |
| Example M | e(s): MOVL MOVL | XAR2, #Array1 XAR3, #Array2 | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 | | |
| Example M | e(s): MOVL MOVL MOV | XAR2, #Array1 XAR3, #Array2 @AR0, #N-1 | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N | | |
| Example M M M | e(s): MOVL MOVL MOV MOV | XAR2, #Array1 XAR3, #Array2 @AR0, #N-1 *, ARP2 | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N ; Set ARP pointer to point to XAR2 | | |
| Example M M M S | e(s): MOVL MOVL MOV NOP SETC | XAR2, #Array1 XAR3, #Array2 @AR0, #N-1 *, ARP2 AMODE | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N ; Set ARP pointer to point to XAR2 ; Make sure AMODE = 1 | | |
| Example M M M S | e(s): MOVL MOVL MOV NOP SETC .lp_a | XAR2, #Array1 XAR3, #Array2 @AR0, #N-1 *, ARP2 AMODE imode | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N ; Set ARP pointer to point to XAR2 ; Make sure AMODE = 1 ; Tell assembler that AMODE = 1 | | |
| Example M M M S Loop: | e(s): MOVL MOVL MOV NOP SETC .lp_a | XAR2, #Array1 XAR2, #Array1 XAR3, #Array2 @AR0, #N-1 *, ARP2 AMODE mode | 32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N ; Set ARP pointer to point to XAR2 ; Make sure AMODE = 1 ; Tell assembler that AMODE = 1 | | |
| Example M M M N S Loop: | e(s): MOVL MOVL MOV NOP SETC .lp_a MOVL | XAR2, #Array1 XAR3, #Array2 @AR0, #N-1 *, ARP2 AMODE mode ACC, *++, ARP3 | <pre>32bitDataAddr(31:0) = XAR(ARP) if(loc16), XAR(ARP) = XAR(ARP) + 1 if(loc32), XAR(ARP) = XAR(ARP) + 2 ; Load XAR2 with start address of Array1 ; Load XAR3 with start address of Array2 ; Load AR0 with loop count N ; Set ARP pointer to point to XAR2 ; Make sure AMODE = 1 ; Tell assembler that AMODE = 1 ; Load ACC with location pointed to by XAR2, ; post-increment XAR2, set ARP to point to XAR3</pre> | | |

XBANZ Loop,*--,ARP2 ; Loop until AR0 == 0, post-decrement AR0, ; set ARP pointer to point to XAR2

| AMODE | "loc16/loc32" Syntax | Description | |
|-------|----------------------|------------------------------------|--|
| Х | * | 32bitDataAddr(31:0) = XAR(ARP) | |
| | | if(loc16), XAR(ARP) = XAR(ARP) + 1 | |
| | | if(loc32), XAR(ARP) = XAR(ARP) + 2 | |

Example(s):

| | | - | | |
|-----|-------|----------------------|---|---|
| | MOVL | XAR2,#Array1+(N-1)*2 | ; | Load XAR2 with end address of Array1 |
| | MOVL | XAR3,#Array2+(N-1)*2 | ; | Load XAR3 with end address of Array2 |
| | MOV | @AR0,#N-1 | ; | Load AR0 with loop count N |
| Loo | p: | | | |
| | NOP | *,ARP2 | ; | Set ARP pointer to point to XAR2 |
| | MOVL | ACC,* | ; | Load ACC with location pointed to by XAR2, |
| | | | ; | post-decrement XAR2 |
| | NOP | *,ARP3 | ; | Set ARP pointer to point to XAR3 |
| | MOVL | *, ACC | ; | Store ACC into location pointed to by XAR3, |
| | | | ; | post-decrement XAR3 |
| | NOP | *,ARP0 | ; | Set ARP pointer to point to XAR0 |
| | XBANZ | Loop,* | ; | Loop until AR0 == 0, post-decrement AR0 |
| | | | | |

| AMODE | "loc16/loc32" Syntax | Description |
|-------|----------------------|------------------------------------|
| 1 | *,ARPn | 32bitDataAddr(31:0) = XAR(ARP) |
| | | if(loc16), XAR(ARP) = XAR(ARP) + 1 |
| | | if(loc32), XAR(ARP) = XAR(ARP) + 2 |
| | | ARP = n |

| | MOVL | XAR2,#Array1+(N-1)*2 | ; | Load XAR2 with end address of Array1 |
|------|--------|----------------------|---|---|
| | MOVL | XAR3,#Array2+(N-1)*2 | ; | Load XAR3 with end address of Array2 |
| | MOV | @AR0,#N-1 | ; | Load AR0 with loop count N |
| | NOP | *,ARP2 | ; | Set ARP pointer to point to XAR2 |
| | SETC | AMODE | ; | Make sure AMODE = 1 |
| | .lp_am | node | ; | Tell assembler that $AMODE = 1$ |
| Loop | : | | | |
| | MOVL | ACC, *, ARP3 | ; | Load ACC with location pointed to by XAR2, |
| | | | ; | post-increment XAR2, set ARP to point |
| | | | ; | to XAR3 |
| | MOVL | *,ACC,ARP0 | ; | Store ACC into location pointed to by XAR3, |
| | | | ; | post-increment XAR3, set ARP to point |
| | | | ; | to XAR0 |
| | XBANZ | Loop,*,ARP2 | ; | Loop until AR0 == 0, post-decrement AR0, |
| | | | ; | set ARP pointer to point to XAR2 |

| | | "leate//acco" 0 | Description |
|----------------------------|--------|-----------------------|--|
| AMODE "loc16/loc32" Syntax | | 10C16/10C32" Syntax | Description |
| Х | | *0++ | 32bitDataAddr(31:0) = XAR(ARP) |
| | | | XAR(ARP) = XAR(ARP) + AR0 |
| | | | Note: The lower 16-bits of XAR0 are added to the selected 32-bit register. Upper 16-bits of XAR0 ignored. AR0 is treated as an unsigned 16-bit value. Overflow into the upper 16-bits of XAR(ARP) can occur. |
| Examp | le(s) |): | |
| | MOVI | L XAR2,#Array1 | ; Load XAR2 with start address of Array1 |
| | MOVI | L XAR3,#Array2 | ; Load XAR3 with start address of Array2 |
| | MOV | @AR0,#4 | ; Set AR0 to copy every fourth value from |
| | | | ; Array1 to Array2 |
| - | MOV | @AR1,#N-1 | ; Load AR1 with loop count N |
| Loop: | NOD | * גםםע | . Set APP pointer to point to YAP? |
| | MOVI | ACC * 0++ | · Load ACC with location pointed to by XAR2 |
| | 110 11 | | ; post-increment XAR2 by AR0 |
| | NOP | *,ARP3 | ; Set ARP pointer to point to XAR3 |
| | MOVI | L *++, ACC | ; Store ACC into location pointed to by XAR3, |
| | | | ; post-increment XAR3 |
| | NOP | *,ARP1 | ; Set ARP pointer to point to XAR1 |
| | XBAI | NZ Loop,* | ; Loop until AR1 == 0, post-decrement AR1 |
| AMODE "loc16/loc32" Syntax | | "loc16/loc32" Syntax | Description |
| 1 | | *0++,ARPn | 32bitDataAddr(31:0) = XAR(ARP) |
| | | | XAR(ARP) = XAR(ARP) + AR0 |
| | | | ARP = n |
| | | | Note: The lower 16-bits of XAB0 are added to the selected 32-bit register Upper |
| | | | 16-bits of XAR0 ignored. AR0 is treated as an unsigned 16-bit value. |
| | | | Overflow into the upper 16-bits of XAR(ARP) can occur. |
| Evamo | le(c) |). | |
| pسمىت | MOVI | , . L XAR2.#Arrav1 | : Load XAR2 with start address of Arrav1 |
| | MOVI | L XAR3,#Arrav2 | ; Load XAR3 with start address of Array2 |
| | MOV | @AR0,#4 | ; Set AR0 to copy every fourth value from |
| | | | ; Array1 to Array2 |
| | MOV | @AR1,#N-1 | ; Load AR1 with loop count N |
| | NOP | *, ARP2 | ; Set ARP pointer to point to XAR2 |
| SET | | | • Make gure $\Delta M \cap DF = 1$ |
| | SEI(| | , Make Suit ANODE - I |

Loop:

| p: | | | |
|----|-------|-----------------|---|
| | MOVL | ACC, *0++, ARP3 | ; Load ACC with location pointed to by XAR2, |
| | | | ; post-increment XAR2 by AR0, set ARP pointer |
| | | | ; to point to XAR3 |
| | MOVL | *++,ACC,ARP1 | ; Store ACC into location pointed to by XAR3, |
| | | | ; post-increment XAR3, set ARP pointer to point |
| | | | ; to XAR1 |
| | XBANZ | Loop,*,ARP2 | ; Loop until AR1 == 0, post-decrement AR1, |
| | | | ; set ARP to point to XAR2 |
| | | | |

| AMODE | | "loc16/loc32" Syntax | Description | | | |
|-------|-------|----------------------|--------------------------------|--|--|--|
| Х | | *0 | 32bitDataAddr(31:0) = XAR(ARP) | | | |
| | | | XAR(A | RP) = XAR(ARP) – AR0 | | |
| | | | Note: | The lower 16-bits of XAR0 are subtracted from the selected 32-bit regis- ter. Upper 16-bits of XAR0 ignored. AR0 is treated as an unsigned 16-bit value. Underflow into the upper 16-bits of XAR(ARP) can occur. | | |
| Examp | le(s) | : | | | | |
| _ | MOVL | XAR2,#Array1+(N- | 1)*8 ; | Load XAR2 with end address of Array1 | | |
| | MOVL | XAR3,#Array2+(N- | 1)*2 ; | Load XAR3 with end address of Array2 | | |
| | MOV | @AR0,#4 | ; | Set AR0 to copy every fourth value from | | |
| | | | ; | Array1 to Array2 | | |
| | MOV | @AR1,#N-1 | ; | Load AR1 with loop count N | | |
| Loop: | | | | | | |
| | NOP | *,ARP2 | ; | Set ARP pointer to point to XAR2 | | |
| | MOVL | ACC,*0 | ; | Load ACC with location pointed to by | | |
| | | | ; | XAR2, post-decrement XAR2 by AR0 | | |
| | NOP | *,ARP3 | ; | Set ARP pointer to point to XAR3 | | |
| | MOVL | *, ACC | ; | Store ACC into location pointed to by | | |
| | | | ; | XAR3, post-decrement XAR3 | | |
| | NOP | *,ARP1 | ; | Set ARP pointer to point to XAR1 | | |
| | XBAN | Z Loop,* | ; | Loop until AR1 == 0, post-decrement AR1 | | |
| | | | | | | |

| AMODE | "loc16/loc32" Syntax | Description | | | | |
|-------|----------------------|---|--|--|--|--|
| 1 | *0,ARPn | 32bitDataAddr(31:0) = XAR(ARP) | | | | |
| | | XAR(ARP) = XAR(ARP) - AR0 | | | | |
| | | ARP = n | | | | |
| | | Note: The lower 16-bits of XAR0 are subtracted from the selected 32-bit register. Upper 16-bits of XAR0 ignored. AR0 is treated as an unsigned 16-bit value. Underflow into the upper 16-bits of XAR(ARP) can occur. | | | | |

Example(s): MOVL XAR2,#Array1+(N-1)*8 ; Load XAR2 with end address of Array1 MOVL XAR3, #Array2+(N-1)*2 ; Load XAR3 with end address of Array2 MOV @AR0,#4 ; Set AR0 to copy every fourth value from ; Array1 to Array2 MOV @AR1,#N-1 ; Load AR1 with loop count N *,ARP2 NOP ; Set ARP pointer to point to XAR2 ; Make sure AMODE = 1 SETC AMODE ; Tell assembler that AMODE = 1 .lp_amode Loop: ; Load ACC with location pointed to by MOVL ACC, *0--, ARP3 ; XAR2, post-decrement XAR2 by AR0, set ARP ; pointer to point to XAR3 MOVL *--, ACC, ARP1 ; Store ACC into location pointed to by ; XAR3, post-decrement XAR3, set ARP ; pointer to point to XAR1 XBANZ Loop, *--, ARP2 ; Loop until AR1 == 0, post-decrement AR1, ; set ARP to point to XAR2

| AMODE | "loc16/loc32" Syntax | Description | | | | |
|-------|----------------------|---|--|--|--|--|
| Х | *BR0++ | 32bitDataAddr(31:0) = XAR(ARP) | | | | |
| | | XAR(ARP)(15:0) = AR(ARP) rcadd AR0 | | | | |
| | | XAR(ARP)(31:16) = unchanged | | | | |
| | | Note: The lower 16-bits of XAR0 are reverse carry added (rcadd) to the lower 16-bits of the selected register. Upper 16-bits of XAR0 ignored. Upper 16-bits of the selected register unchanged by the operation. | | | | |

```
Example(s):
```

| | (~) . | | | |
|--------|---------|-------------------|---|--|
| ; Trai | nsfer (| contents of Array | 1 | to Array2 in bit reverse order: |
| | MOVL | XAR2,#Array1 | ; | Load XAR2 with start address of Array1 |
| | MOVL | XAR3,#Array2 | ; | Load XAR3 with start address of Array2 |
| | MOV | @AR0,#N | ; | Load AR0 with size of array, |
| | | | ; | N must be a multiple of 2 (2,4,8,16,) |
| | MOV | @AR1,#N-1 | ; | Load AR1 with loop count N |
| Loop: | | | | |
| | NOP | *,ARP2 | ; | Set ARP pointer to point to XAR2 |
| | MOVL | ACC, *++ | ; | Load ACC with location pointed to by XAR2, |
| | | | ; | post-increment XAR2 |
| | NOP | *,ARP3 | ; | Set ARP pointer to point to XAR3 |
| | MOVL | *BR0++,ACC | ; | Store ACC into location pointed to by XAR3, |
| | | | ; | post-increment XAR3 with AR0 reverse carry add |
| | NOP | *,ARP1 | ; | Set ARP pointer to point to XAR1 |
| | XBANZ | Loop,* | ; | Loop until AR1 == 0, post-decrement AR1 |
| | | | | |

| AMODE | "loc16/loc32" Syntax | Description | | | | |
|-------|----------------------|---|--|--|--|--|
| 1 | *BR0++,ARPn | 32bitDataAddr(31:0) = XAR(ARP) | | | | |
| | | XAR(ARP)(15:0) = AR(ARP) rcadd AR0 | | | | |
| | | XAR(ARP)(31:16) = unchanged | | | | |
| | | ARP = n | | | | |
| | | Note: The lower 16-bits of XAR0 are reverse carry added (rcadd) to the lower 16-bits of the selected register. Upper 16-bits of XAR0 ignored. Upper 16-bits of the selected register unchanged by the operation. | | | | |

Example(s):

| 1. AC | impre(b). | | |
|-------|-----------|--------------------|---|
| ; 1 | ransfer | contents of Array1 | to Array2 in bit reverse order: |
| | MOVL | XAR2,#Array1 ; | Load XAR2 with start address of Array1 |
| | MOVL | XAR3,#Array2 ; | Load XAR3 with start address of Array2 |
| | MOV | @AR0,#N ; | Load AR0 with size of array, |
| | | ; | N must be a multiple of 2 (2,4,8,16,) |
| | MOV | @AR1,#N-1 ; | Load AR1 with loop count N |
| | NOP | *,ARP2 ; | Set ARP pointer to point to XAR2 |
| | SETC | AMODE ; | Make sure AMODE = 1 |
| | .lp a | amode ; | Tell assembler that $AMODE = 1$ |
| Loc | ; q | | |
| | MOVL | ACC, *++, ARP3 ; | Load ACC with location pointed to by XAR2, |
| | | ; | post-increment XAR2, set ARP pointer to point |
| | | ; | to XAR3 |

| MOVL | *BR0++,ACC,ARP1 | ; | Store ACC into location pointed to by XAR3, |
|-------|-----------------|---|---|
| | | ; | post-increment XAR3 with AR0 reverse carry |
| | | ; | add, set ARP pointer to point to XAR1 |
| XBANZ | Loop,*,ARP2 | ; | Loop until AR1 == 0, post-decrement AR1, |
| | | ; | set ARP to point to XAR2 |

| AMODE | "loc16/loc32" Syntax | Description | | | | |
|-------|----------------------|---|--|--|--|--|
| Х | *BR0 | Address Generation: | | | | |
| | | 32bitDataAddr(31:0) = XAR(ARP) | | | | |
| | | XAR(ARP)(15:0) = AR(ARP) rbsub AR0 {see note [1]} | | | | |
| | | XAR(ARP)(31:16) = unchanged | | | | |
| | | Note: The lower 16-bits of XAR0 are reverse borrow subtracted (rbsub) from the lower 16-bits of the selected register. Upper 16-bits of XAR0 ignored. Upper 16-bits of the selected register unchanged by the operation. | | | | |

Example(s):

| ; Tr | ansfer | contents of Array1 to | A | rray2 in bit reverse order: |
|------|--------|-----------------------|---|---|
| | MOVL | XAR2,#Array1+(N-1)*2 | ; | Load XAR2 with end address of Array1 |
| | MOVL | XAR3,#Array2+(N-1)*2 | ; | Load XAR3 with end address of Array2 |
| | MOV | @AR0,#N | ; | Load AR0 with size of array, |
| | | | ; | N must be a multiple of 2 (2,4,8,16,) |
| | MOV | @AR1,#N-1 | ; | Load AR1 with loop count N |
| Loop | : | | | |
| | NOP | *,ARP2 | ; | Set ARP pointer to point to XAR2 |
| | MOVL | ACC,* | ; | Load ACC with location pointed to by |
| | | | ; | XAR2, post-decrement XAR2 |
| | NOP | *,ARP3 | ; | Set ARP pointer to point to XAR3 |
| | MOVL | *BR0, ACC | ; | Store ACC into location pointed to by |
| | | | ; | XAR3, post-decrement XAR3 with AR0 |
| | | | ; | reverse borrow subtract |
| | NOP | *,ARP1 | ; | Set ARP pointer to point to XAR1 |
| | XBANZ | Loop,* | ; | Loop until AR1 == 0, post-decrement AR1 |

| AMODE | "loc16/loc32" Syntax | Description | | | | |
|-------|----------------------|---|--|--|--|--|
| 1 | *BR0,ARPn | 32bitDataAddr(31:0) = XAR(ARP) | | | | |
| | | XAR(ARP)(15:0) = AR(ARP) rbsub AR0 | | | | |
| | | XAR(ARP)(31:16) = unchanged | | | | |
| | | ARP = n | | | | |
| | | Note: The lower 16-bits of XAR0 are reverse borrow subtracted (rbsub) from the lower 16-bits of the selected register. Upper 16-bits of XAR0 ignored. Upper 16-bits of the selected register unchanged by the operation. | | | | |

```
Example(s):
```

| | ± , , | | | |
|------|--------|-----------------------|---|--|
| ; T1 | ansfer | contents of Array1 to | A | rray2 in bit reverse order: |
| | MOVL | XAR2,#Array1+(N-1)*2 | ; | Load XAR2 with end address of Array1 |
| | MOVL | XAR3,#Array2+(N-1)*2 | ; | Load XAR3 with end address of Array2 |
| | MOV | @AR0,#N | ; | Load AR0 with size of array, |
| | | | ; | N must be a multiple of 2 (2,4,8,16,) |
| | MOV | @AR1,#N-1 | ; | Load AR1 with loop count N |
| | NOP | *,ARP2 | ; | Set ARP pointer to point to XAR2 |
| | SETC | AMODE | ; | Make sure AMODE = 1 |
| | .lp_a | mode | ; | Tell assembler that $AMODE = 1$ |
| Loop |): | | | |
| | MOVL | ACC, *, ARP3 | ; | Load ACC with location pointed to by |
| | | | ; | XAR2, post-decrement XAR2, set ARP |
| | | | ; | pointer to point to XAR3 |
| | MOVL | *BR0,ACC,ARP1 | ; | Store ACC into location pointed to by |
| | | | ; | XAR3, post-decrement XAR3 with AR0 |
| | | | ; | reverse borrow subtract, set ARP pointer |
| | | | ; | to point to XAR1 |
| | XBANZ | Loop,*,ARP2 | ; | Loop until AR1 == 0, post-decrement AR1, |
| | | | ; | set ARP pointer to point to XAR2 |

Reverse carry addition or reverse carry subtraction is used to implement bitreversed addressing as used in the re-ordering of data elements in FFT algorithms. Typically, AR0 is initialized with the (FFT size) /2. The value of AR0 is then added or subtracted, with reverse carry addition or subtraction, to generate the bit reversed address:

Reverse Carry Addition Example Is Shown Below (FFT size = 16):

| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 0000 |
|----------------|---|------|------|------|------|
| + AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 1000 |
| + AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 0100 |
| + AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 1100 |
| + AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 0010 |
| + AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 1010 |
| | | | | | |

Reverse Borrow Subtraction Example Is Shown Below (FFT size = 16):

| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 0000 |
|----------------|---|------|------|------|------|
| - AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 1111 |
| - AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 0111 |
| - AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 1011 |
| - AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 0011 |
| - AR0 | = | 0000 | 0000 | 0000 | 1000 |
| | - | | | | |
| XAR(ARP)(15:0) | = | 0000 | 0000 | 0000 | 1101 |
| | | | | | |
| | | | | | |

On the C28x, the bit reversed addressing is restricted to block size < 64K. This is OK since most FFT implementations are much less than this.
| AMODE | "loc16/loc32" Syntax | Description | |
|-------|----------------------|----------------------------------|--|
| 0 | *AR6%++ | 32bitDataAddr(31:0) = XAR6 | |
| | | if(XAR6(7:0) == XAR1(7:0)) | |
| | | { | |
| | | XAR6(7:0) = 0x00 | |
| | | XAR6(15:8) = unchanged | |
| | | } | |
| | | else | |
| | | { | |
| | | if(16-bit data), XAR6(15:0) =+ 1 | |
| | | if(32-bit data), XAR6(15:0) =+ 2 | |
| | | } | |
| | | XAR6(31:16) = unchanged | |
| | | ARP = 6 | |

5.6.3 Circular Indirect Addressing Modes (XAR6, XAR1)

As seen in Figure 5–1, buffer size is determined by the 8 LSBs of AR1 or AR1[7:0]. Specifically, the buffer size is AR1[7:0] +1. When AR1[7:0] is 255, then the buffer size is at its maximum size of 256 words.

XAR6 points to the current address in the buffer. The top of the buffer must be at an address where the 8 LSBs are all 0s.

If one of the instructions accessing the circular buffer performs a 32-bit operation, make sure XAR6 and AR1 are both even before the buffer is accessed.

Figure 5–1. Circular Buffer with AMODE = 0



| AMODE | "loc16/loc32" Syntax | Description | |
|-------|----------------------|---|--|
| 1 | *+XAR6[AR1%++] | 32bitDataAddr(31:0) = XAR6 + AR1 | |
| | | if(XAR1(15:0) == XAR1(31:16)) | |
| | | { | |
| | | XAR1(15:0) = 0x0000 | |
| | | } | |
| | | else | |
| | | { | |
| | | if(16-bit data), XAR1(15:0) =+ 1 | |
| | | if(32-bit data), XAR1(15:0) =+ 2 | |
| | | } | |
| | | XAR1(31:16) = unchanged | |
| | | ARP = 6 | |
| | | Note: With this addressing mode, there is no circular buffer alignment requirements. | |

As seen in Figure 5–2, buffer size is determined by the upper 16 bits of XAR1 or XAR1[31:16]. Specifically, the size is XAR1[31:16] + 1.

XAR6 points to the top of the buffer.

The current address in the buffer is pointed to by XAR6 with an offset of XAR1[15:0].

If the instructions that access the circular buffer perform 32-bit operations, make sure XAR6 and XAR1[31:16] are even.

Figure 5–2. Circular Buffer with AMODE = 1



| Εz | <pre>kample(s):</pre> | | | |
|----|-----------------------|--------------------------|------|---|
| ; | Calculate | FIR filter (X[N] = data | arra | ay, C[N] = coefficientv array): |
| | MOVW | DP,#Xindex | ; | Load DP with page address of Xindex |
| | MOVL | XAR6,#X | ; | Load XAR6 with start address of X array |
| | MOV | @AH,#N | ; | Load AH with size of array X (N) |
| | MOV | AL,@Xindex | ; | Load AL with current circular index |
| | MOVL | XAR1,@ACC | ; | Load parameters into XAR1 |
| | MOVL | XAR7,#C | ; | Load XAR7 with start address of C array |
| | SPM | -4 | ; | Set product shift mode to ">> 4" |
| | ZAPA | | ; | Zero ACC, P, OVC |
| | RPT | #N-1 | ; | Repeat next instruction N times |
| | QMACL | P,*+XAR6[AR1%++],*XAR7++ | ; | ACC = ACC + P >> 4, |
| | | | ; | P = (*AR6%++ * *XAR7++) >> 32 |
| | ADDL | ACC,P << PM | ; | Final accumulate |
| | MOV | @Xindex,AR1 | ; | Store AR1 into current X index |
| | MOVL | @Sum,ACC | ; | Store result into sum |

5.7 Register Addressing Modes

This section includes register addressing modes for 32-bit and 16-bit registers.

5.7.1 32-Bit Register Addressing Modes

| AMODE | "loc32" Syntax | Description |
|-------|----------------|---|
| Х | @ACC | Access contents of 32-bit ACC register. |
| | | When the "@ACC" register is the destination operand, this may affect the Z,N,V,C,OVC flags. |

Example(s):

| mpro (b | • | |
|---------|-----------|---|
| MOVL | XAR6,@ACC | ; Load XAR6 with contents of ACC |
| MOVL | @ACC,XT | ; Load ACC with contents of XT register |
| ADDL | ACC,@ACC | ; $ACC = ACC + ACC$ |

| AMODE | "loc32" Syntax | Description | |
|------------|----------------|---------------------------------------|--|
| Х | @P | Access contents of 32-bit P register. | |
| Example(s) |): | | |
| MOVL | XAR6,@P ; | Load XAR6 with contents of P | |
| MOVL | @P,XT ; | Load P with contents of XT register | |
| ADDL | ACC,@P ; | ACC = ACC + P | |
| | | | |

| AMODE | "loc32" Syntax | Description | |
|------------------------------------|--|--|--|
| Х | @XT | Access contents of 32-bit XT register. | |
| Example(s) MOVL MOVL ADDL |): XAR6,@XT ; P,@XT ; ACC,@XT ; | Load XAR6 with contents of XT Load P with contents of XT register ACC = ACC + XT | |

| AMODE | "loc32" Syntax | Description |
|-----------------------------------|--|--|
| Х | @XARn | Access contents of 32-bit XARn registers. |
| Example(s MOVL MOVL ADDL |): XAR6,@XAR2 ; P,@XAR2 ; ACC,@XAR2 ; | Load XAR6 with contents of XAR2 Load P with contents of XAR2 register ACC = ACC + XAR2 |

Note: When writing assembly code, the "@" symbol in front of the register is optional. For example: "MOVL ACC,@P" or "MOVL ACC,P". The disassembler will use the @ to indicate operands that are "loc16" or "loc32". For example, MOVL ACC, @P is the MOVL ACC, loc32 instruction and MOVL @ACC, P is the MOVL loc32, P instruction.

5.7.2 16-Bit Register Addressing Modes

| AMODE | "loc16" Syntax | Description | |
|-------|----------------|--|--|
| Х | @AL | Access contents of 16-bit AL register. | |
| | | AH register contents are un-affected. | |
| | | When the "@AL" register is the destination operand, this may affect the Z,N,V,C,OVC flags. | |

Example(s):

| p ± 0 (0) | , • | | |
|-------------|--------|---|-----------------------------|
| MOV | PH,@AL | ; | Load PH with contents of AL |
| ADD | AH,@AL | ; | AH = AH + AL |
| MOV | T,@AL | ; | Load T with contents of AL |

| AMODE | "loc16" Syntax | Description |
|-------|----------------|--|
| Х | @AH | Access contents of 16-bit AH register. |
| | | AL register contents are un-affected. |
| | | When the "@AH" register is the destination operand, this may affect the Z,N,V,C,OVC flags. |

Example(s): MOV PI

| MOV | PH,@AH | ; | Load PH with contents of AH |
|-----|--------|---|-----------------------------|
| ADD | AL,@AH | ; | AL = AL + AH |
| MOV | Т,@АН | ; | Load T with contents of AH |

| AMODE | "loc16" Syntax | Description | |
|-------|----------------|--|--|
| Х | @PL | Access contents of 16-bit PL register. | |
| | | PH register contents are un-affected. | |

Example(s):

| T () | | | | | | | | |
|------|--------|---|------|----|------|----------|------|----|
| MOV | PH,@PL | ; | Load | PH | with | contents | s of | PL |
| ADD | AL,@PL | ; | AL = | AL | + PI | ı | | |
| MOV | T,@PL | ; | Load | Т | with | contents | of | PL |

| AMODE | "loc16" Syntax | | Description |
|------------|----------------|---|--|
| Х | @PH | | Access contents of 16-bit PH register. |
| | | | PL register contents are un-affected. |
| Example(s) |): | | |
| MOV | PL,@PH | ; | Load PL with contents of PH |
| ADD | AL,@PH | ; | AL = AL + PH |
| MOV | Т,@РН | ; | Load T with contents of PH |

| ; | Load | PL | with | contents | s of | PH |
|---|------|---------------------|------|----------|------|----|
| ; | AL = | AL | + PH | | | |
| ; | Load | τт | with | contents | of | PH |

| AMODE | "loc16" Syntax | Description |
|-----------|----------------|--|
| Х | @TH | Access contents of 16-bit TH register. |
| | | TL register contents are unaffected. |
| Example(s |): | |

| ± , , | | | |
|-------|--------|---|---|
| MOV | PL,@T | ; | ; Load PL with contents of T |
| ADD | AL,@T | ; | ; $AL = AL + T$ |
| MOVZ | AR4,@T | ; | : Load AR4 with contents of T, AR4H = 0 |

| AMODE | "loc16" Syntax | Description |
|------------|----------------|--|
| х | @SP | Access contents of 16-bit SP register. |
| Example(s) |): | |
| MOVZ | AR4,@SP ; | Load AR4 with contents of SP, AR4H = 0 |
| MOV | AL,@SP ; | Load AL with contents of SP |
| MOV | @SP,AH ; | Load SP with contents of AH |

| AMODE | "loc16" Syntax | Description |
|-----------|--|--|
| Х | X @ARn Access contents of 16-bit AR0 to AR7 registers. | |
| | | AR0H to AR7H register contents are unaffected. |
| Example(s |): | |
| MOVZ | AR4,@AR2 ; | Load AR4 with contents of AR2, AR4H = 0 |
| MOV | AL,@AR3 ; | Load AL with contents of AR3 |
| MOV | @AR5,AH ; | Load AR5 with contents of AH, AR5H = unchanged |

5.8 Data/Program/IO Space Immediate Addressing Modes

| Syntax | Description |
|------------|---|
| *(0:16bit) | <pre>32BitDataAddr(31:16) = 0 32BitDataAddr(15:0) = 16-bit immediate value Note: If instruction is repeated, the address is post-incremented on each iteration. This addressing mode can only access the low 64K of data space.</pre> |

Instructions that use this addressing mode: MOV loc16,*(0:16bit) ; [loc16] = [0:16bit] MOV *(0:16bit),loc16 ; [loc16] = [0:16bit]

| Syntax | Description |
|--------|---|
| *(PA) | 32BitDataAddr(31:16) = 0 32BitDataAddr(15:0) = PA 16-bit immediate value Note: If instruction is repeated, the address is post-incremented on each iteration. The I/O strobe signal is toggled when accessing I/O space with this addressing mode. The data space address lines are used for accessing I/O space. |

| Instru | ctions that use | this | addressing mode: | |
|--------|-----------------|------|--------------------------|--------------------|
| OUT | *(PA),loc16 | | ; IOspace[0:PA] = [lc | oc16] |
| UOUT | *(PA),loc16 | | ; IOspace[0:PA] = [lc | c16] (unprotected) |
| IN | loc16,*(PA) | | ; $[loc16] = IOspace[0:$ | PA] |

| Syntax | Description |
|--------|--|
| 0:pma | <pre>22BitProgAddr(21:16) = 0 22BitProgAddr(15:0) = pma 16-bit immediate value Note: If instruction is repeated, the address is post-incremented on each iteration. This addressing mode can only access the low 64K of program space.</pre> |

Instructions that use this addressing mode: MAC P,loc16,0:pma ; ACC = ACC + P << PM, ; P = [loc16] * ProgSpace[0:pma]

| Syntax | Description |
|-------------------|---|
| *(pma) | <pre>22BitProgAddr(21:16) = 0x3F 22BitProgAddr(15:0) = pma 16-bit immediate value Note: If instruction is repeated, the address is post-incremented on each iteration. This addressing mode can only access the upper 64K of program space.</pre> |
| Instructions that | ugo thig addrogging mode. |

| Instruct | LIONS LHAL USE LHIS | addressing mode: |
|----------|---------------------|--------------------------------------|
| XPREAD | loc16,*(pma) | ; [loc16] = ProgSpace[0x3F:pma] |
| XMAC | P,loc16,*(pma) | ; ACC = ACC + $P \ll PM$, |
| | | ; P = [loc16] * ProgSpace[0x3F:pma] |
| XMACD | P,loc16,*(pma) | ; ACC = ACC + $P \ll PM$, |
| | | ; P = [loc16] * ProgSpace[0x3F:pma], |
| | | ; $[loc16+1] = [loc16]$ |

5.9 Program Space Indirect Addressing Modes

| Syntax | Description |
|--------|---|
| *AL | 22BitProgAddr(21:16) = 0x3F 22BitProgAddr(15:0) = AL |
| | Note: If instruction is repeated, the address in AL is copied to a shadow register and the value post–incremented on each iteration. The AL register is not modified. This addressing mode can only access the upper 64K of program space. |

Instructions that use this addressing mode: XPREAD loc16,*AL ; [loc16] = ProgSpace[0x3F:AL] XPWRITE *AL,loc16 ; ProgSpace[0x3F:AL] = [loc16]

| Syntax | Description |
|-----------------------|--|
| *XAR7 | 22BitProgAddr(21:0) = XAR7 |
| | Note: If instruction is repeated, only in the XPREAD and XPWRITE instructions, is the address contained in XAR7 copied to a shadow register and the value post–incremented on each iteration. The XAR7 register is not modified. For all other instructions, the address is not incremented even when repeated. |
| Instructions that use | this addressing mode: |

| Insciu | cions chac use chi | Ð | addressing mode. |
|--------|--------------------|---|---|
| MAC | P,loc16,*XAR7 | ; | $ACC = ACC + P \ll PM$, |
| | | ; | P = [loc16] * ProgSpace[*XAR7] |
| DMAC | ACC:P,loc32,*XAR7 | ; | ACC = ([loc32].MSW * ProgSpace[*XAR7].MSW) >> PM, |
| | | ; | P = ([loc32].LSW * ProgSpace[*XAR7].MSW) >> PM |
| QMACL | P,loc32,*XAR7 | ; | ACC = ACC + P >> PM, |
| | | ; | P = ([loc32] * ProgSpace[*XAR7]) >> 32 |
| IMACL | P,loc32,*XAR7 | ; | ACC = ACC + P, |
| | | ; | P = ([loc32] * ProgSpace[*XAR7]) << PM |
| PREAD | loc16,*XAR7 | ; | <pre>[loc16] = ProgSpace[*XAR7]</pre> |
| PWRITE | *XAR7,loc16 | ; | <pre>ProgSpace[*XAR7] = [loc16]</pre> |
| | | | |

| Syntax | Description |
|---------|--|
| *XAR7++ | <pre>22BitProgAddr(21:0) = XAR7, if(16-bit operation) XAR7 = XAR7 + 1, if(32-bit operation) XAR7 = XAR7 + 2</pre> Note: If instruction is repeated, the address is post-incremented as normal. |

Instructions that use this addressing mode: MAC P,loc16,*XAR7++ ; ACC = ACC + P << PM, ; P = [loc16] * ProgSpace[*XAR7++] DMAC ACC:P,loc32,*XAR7++ ; ACC=([loc32].MSW * ProgSpace[*XAR7++].MSW)>>PM, ; P=([loc32].LSW * ProgSpace[*XAR7++].MSW)>>PM QMACL P,loc32,*XAR7++ ; ACC = ACC + P >> PM, ; P = ([loc32] * ProgSpace[*XAR7++]) >> 32 IMACL P,loc32,*XAR7++ ; ACC = ACC + P, ; P = ([loc32] * ProgSpace[*XAR7++]) << PM</pre>

5.10 Byte Addressing Modes

| Syntax | Description |
|-----------------------------|---|
| *+XARn[AR0] | 32BitDataAddr(31:0) = XARn + Offset (Offset = |
| *+XARn[AR1] *+XARn[3bit] | <pre>AR0/AR1/3bit) if(Offset == Even Value) Access LSByte Of 16-bit Memory Location;</pre> |
| | Leave MSByte untouched; if(Offset == Odd Value) Access MSByte Of 16-bit Memory Location; Leave LSByte untouched; |
| | Note: For all other addressing modes, only the LSByte of the addressed location is accessed, the MSByte is left untouched. |

```
Instructions that use this addressing mode:
MOVB AX.LSB,loc16 ; if( address mode == *+XARn[AR0/AR1/3bit] )
                         if ( offset == even )
                     ;
                            AX.LSB = [loc16].LSB;
                     ;
                            AX.MSB = 0x00;
                     ;
                          if( offset == odd )
                     ;
                            AX.LSB = [loc16].MSB;
                      ;
                             AX.MSB = 0x00;
                      ;
                     ; else
                     ;
                          AX.LSB = [loc16].LSB;
                          AX.MSB = 0x00;
                     ;
                    ; if( address mode == *+XARn[AR0/AR1/3bit] )
MOVB AX.MSB, loc16
                     ; if( offset == even )
                            AX.LSB = untouched;
                     ;
                             AX.MSB = [loc16].LSB;
                     ;
                         if( offset == odd )
                     ;
                            AX.LSB = untouched;
                     ;
                             AX.MSB = [loc16].MSB;
                     ;
                     ; else
                          AX.LSB = untouched;
                     ;
                          AX.MSB = [loc16].LSB;
                     ;
MOVB loc16,AX.LSB
                    ; if ( address mode == *+XARn[AR0/AR1/3bit] )
                         if( offset == even )
                     ;
                             [loc16].LSB = AX.LSB
                     ;
                             [loc16].MSB = untouched;
                     ;
                         if( offset == odd )
                     ;
                            [loc16].LSB = untouched;
                     ;
                             [loc16].MSB = AX.LSB;
                     ;
                     ; else
                          [loc16].LSB = AX.LSB;
                     ;
                          [loc16].MSB = untouched;
                     ;
                    ; if( address mode == *+XARn[AR0/AR1/3bit] )
MOVB loc16,AX.MSB
                         if ( offset == even )
                     ;
                             [loc16].LSB = AX.MSB
                     ;
                             [loc16].MSB = untouched;
                     ;
                         if( offset == odd )
                     ;
                             [loc16].LSB = untouched;
                     ;
                             [loc16].MSB = AX.MSB;
                     ;
                     ; else
                          [loc16].LSB = AX.MSB;
                     ;
                         [loc16].MSB = untouched;
                     ;
```

5.11 Alignment of 32-Bit Operations

All 32-bit reads and writes to memory are aligned at the memory interface to an even address boundary with the least significant word of the 32-bit data aligned to the even address. The output of the address generation unit does not force alignment, hence pointer values retain their values. For example:

```
MOVB AR0,#5; AR0 = 5
MOVL *AR0,ACC ; AL -> address 0x000004
; AH -> address 0x000005
; AR0 = 5
```

The programmer must take the above into account when generating addresses that are not aligned to an even boundary.

32-bit operands are stored in the following order; low order bits, 0 to 15, followed by the high order bits, 16 to 31, on the next highest 16-bit address increment (little-endian format).

Chapter 6

C28x Assembly Language Instructions

This chapter presents summaries of the instruction set, defines special symbols and notations used, and describes each instruction in detail in alphabetical order.

| | Page |
|---|---|
| Instruction Set Summary (Organized by Function) | 6-2 |
| Register Operations | 6-4 |
| | Instruction Set Summary (Organized by Function) |

6.1 Instruction Set Summary (Organized by Function)

Note: The examples in this chapter assume that the device is already operating in C28x Mode (OBJMODE == 1, AMODE == 0). To put the device into C28x mode following a reset, you must first set the OBJMODE bit in ST1 by executing the "C28OBJ" (or "SETC OBJMODE") instruction.

| Symbol | Description |
|-----------|---|
| XARn | XAR0 to XAR7 registers |
| ARn, ARm | Lower 16-bits of XAR0 to XAR7 registers |
| ARnH | Upper 16-bits of XAR0 to XAR7 registers |
| ARPn | 3-bit auxiliary register pointer, ARP0 to ARP7 |
| | ARP0 points to XAR0 and ARP7 points to XAR7 |
| AR(ARP) | Lower 16-bits of auxiliary register pointed to by ARP |
| XAR(ARP) | Auxiliary registers pointed to by ARP |
| AX | Accumulator high (AH) and low (AL) registers |
| # | Immediate operand |
| PM | Product shift mode (+4,1,0,-1,-2,-3,-4,-5,-6) |
| PC | Program counter |
| ~ | Bitwise compliment |
| [loc16] | Contents of 16-bit location |
| 0:[loc16] | Contents of 16-bit location, zero extended |
| S:[loc16] | Contents of 16-bit location, sign extended |
| [loc32] | Contents of 32-bit location |
| 0:[loc32] | Contents of 32-bit location, zero extended |
| S:[loc32] | Contents of 32-bit location, sign extended |
| 7bit | 7-bit immediate value |
| 0:7bit | 7-bit immediate value, zero extended |
| S:7bit | 7-bit immediate value, sign extended |
| 8bit | 8-bit immediate value |
| 0:8bit | 8-bit immediate value, zero extended |

Table 6–1. Instruction Set Summary (Organized by Function)

| Symbol | Description |
|---------|--|
| S:8bit | 8-bit immediate value, sign extended |
| 10bit | 10-bit immediate value |
| 0:10bit | 10-bit immediate value, zero extended |
| 16bit | 16-bit immediate value |
| 0:16bit | 16-bit immediate value, zero extended |
| S:16bit | 16-bit immediate value, sign extended |
| 22bit | 22-bit immediate value |
| 0:22bit | 22-bit immediate value, zero extended |
| LSb | Least Significant bit |
| LSB | Least Significant Byte |
| LSW | Least Significant Word |
| MSb | Most Significant bit |
| MSB | Most Significant Byte |
| MSW | Most Significant Word |
| OBJ | OBJMODE bit state for which instruction is valid |
| Ν | Repeat count (N = 0,1,2,3,4,5,6,7,) |
| {} | Optional field |
| = | Assignment |
| == | Equivalent to |

Table 6–1. Instruction Set Summary (Organized by Function) (Continued)

6.2 Register Operations

Note:

The examples in this chapter assume that the device is already operating in C28x Mode (OBJMODE == 1, AMODE == 0). To put the device into C28x mode following a reset, you must first set the OBJMODE bit in ST1 by executing the "C28OBJ" (or "SETC OBJMODE") instruction.

| Mnemonic | : | Description | Page | | |
|--------------------------------------|---------------|---|-------|--|--|
| XARn Register Operations (XAR0-XAR7) | | | | | |
| ADDB | XARn,#7bit | Add 7-bit constant to auxiliary register | 6-33 | | |
| ADRK | #8bit | Add 8-bit constant to current auxiliary register | 6-42 | | |
| CMPR | 0/1/2/3 | Compare auxiliary registers | 6-82 | | |
| MOV | AR6/7,loc16 | Load auxiliary register | 6-160 | | |
| MOV | loc16,ARn | Store 16-bit auxiliary register | 6-168 | | |
| MOV | XARn,PC | Save the current program counter | 6-182 | | |
| MOVB | XARn,#8bit | Load auxiliary register with 8-bit value | 6-200 | | |
| MOVB | AR6/7,#8bit | Load auxiliary register with an 8-bit constant | 6-188 | | |
| MOVL | XARn,loc32 | Load 32-bit auxiliary register | 6-214 | | |
| MOVL | loc32,XARn | Store 32-bit auxiliary register | 6-210 | | |
| MOVL | XARn,#22bit | Load 32-bit auxiliary register with constant value | 6-215 | | |
| MOVZ | ARn,loc16 | Load lower half of XARn and clear upper half | 6-225 | | |
| SBRK | #8bit | Subtract 8-bit constant from current auxiliary register | 6-319 | | |
| SUBB | XARn,#7bit | Subtract 7-bit constant from auxiliary register | 6-342 | | |
| DP Registe | er Operations | | | | |
| MOV | DP,#10bit | Load data-page pointer | 6-162 | | |
| MOVW | DP,#16bit | Load the entire data page | 6-223 | | |
| MOVZ | DP,#10bit | Load data page and clear high bits | 6-226 | | |
| SP Registe | er Operations | | | | |
| ADDB | SP,#7bit | Add 7-bit constant to stack pointer | 6-32 | | |
| POP | ACC | Pop ACC register from stack | 6-267 | | |
| POP | AR1:AR0 | Pop AR1 & AR0 registers from stack | 6-268 | | |
| POP | AR1H:AR0H | Pop AR1H & AR0H registers from stack | 6-269 | | |

Table 6–2. Register Operations

| Mnemonic | | Description | Page | | |
|------------------------------------|-----------|-------------------------------------|-------|--|--|
| SP Register Operations (Continued) | | | | | |
| POP | AR3:AR2 | Pop AR3 & AR2 registers from stack | 6-268 | | |
| POP | AR5:AR4 | Pop AR5 & AR4 registers from stack | 6-268 | | |
| POP | DBGIER | Pop DBGIER register from stack | 6-270 | | |
| POP | DP:ST1 | Pop DP & ST1 registers on stack | 6-272 | | |
| POP | DP | Pop DP register from stack | 6-271 | | |
| POP | IFR | Pop IFR register from stack | 6-273 | | |
| POP | loc16 | Pop "loc16" data from stack | 6-274 | | |
| POP | Р | Pop P register from stack | 6-275 | | |
| POP | RPC | Pop RPC register from stack | 6-276 | | |
| POP | ST0 | Pop ST0 register from stack | 6-277 | | |
| POP | ST1 | Pop ST1 register from stack | 6-278 | | |
| POP | T:ST0 | Pop T & ST0 registers from stack | 6-279 | | |
| POP | ХТ | Pop XT register from stack | 6-281 | | |
| POP | XARn | Pop auxiliary register from stack | 6-280 | | |
| PUSH | ACC | Push ACC register on stack | 6-284 | | |
| PUSH | ARn:ARn | Push ARn & ARn registers on stack | 6-285 | | |
| PUSH | AR1H:AR0H | Push AR1H & AR0H registers on stack | 6-286 | | |
| PUSH | DBGIER | Push DBGIER register on stack | 6-287 | | |
| PUSH | DP:ST1 | Push DP & ST1 registers on stack | 6-289 | | |
| PUSH | DP | Push DP register on stack | 6-288 | | |
| PUSH | IFR | Push IFR register on stack | 6-290 | | |
| PUSH | loc16 | Push "loc16" data on stack | 6-291 | | |
| PUSH | Р | Push P register on stack | 6-292 | | |
| PUSH | RPC | Push RPC register on stack | 6-293 | | |
| PUSH | ST0 | Push ST0 register on stack | 6-294 | | |
| PUSH | ST1 | Push ST1 register on stack | 6-295 | | |

Table 6-2. Register Operations (Continued)

| Mnemonic | | Description | Page | |
|------------------------------------|---------------------|--|-------|--|
| SP Register Operations (Continued) | | | | |
| PUSH | T:ST0 | Push T & ST0 registers on stack | 6-296 | |
| PUSH | XT | Push XT register on stack | 6-298 | |
| PUSH | XARn | Push auxiliary register on stack | 6-297 | |
| SUBB | SP,#7bit | Subtract 7-bit constant from the stack pointer | 6-341 | |
| AX Register | Operations (AH, AL) | | | |
| ADD | AX,loc16 | Add value to AX | 6-27 | |
| ADD | loc16,AX | Add AX to specified location | 6-28 | |
| ADDB | AX,#8bit | Add 8-bit constant to AX | 6-31 | |
| AND | AX,loc16,#16bit | Bitwise AND | 6-45 | |
| AND | AX,loc16 | Bitwise AND | 6-49 | |
| AND | loc16,AX | Bitwise AND | 6-48 | |
| ANDB | AX,#8bit | Bitwise AND 8-bit value | 6-51 | |
| ASR | AX,116 | Arithmetic shift right | 6-53 | |
| ASR | AX,T | Arithmetic shift right by $T(3:0) = 015$ | 6-54 | |
| CMP | AX,loc16 | Compare | 6-74 | |
| CMPB | AX,#8bit | Compare 8-bit value | 6-79 | |
| FLIP | AX | Flip order of bits in AX register | 6-96 | |
| LSL | AX,116 | Logical shift left | 6-135 | |
| LSL | AX,T | Logical shift left by T(3:0) = 015 | 6-136 | |
| LSR | AX,116 | Logical shift right | 6-140 | |
| LSR | AX,T | Logical shift right by T(3:0) = 015 | 6-136 | |
| MAX | AX,loc16 | Find the maximum | 6-149 | |
| MIN | AX,loc16 | Find the minimum | 6-153 | |
| MOV | AX,loc16 | Load AX | 6-161 | |
| MOV | loc16,AX | Store AX | 6-169 | |
| MOV | loc16,AX,COND | Store AX register conditionally | 6-170 | |

Table 6–2. Register Operations (Continued)

| Mnemonic | | Description | Page | |
|---|---------------------|--|-------|--|
| AX Register Operations (AH, AL) (Continued) | | | | |
| MOVB | AX,#8bit | Load AX with 8-bit constant | 6-189 | |
| MOVB | AX.LSB,loc16 | Load LSB of AX reg, MSB = 0x00 | 6-190 | |
| MOVB | AX.MSB,loc16 | Load MSB of AX reg, LSB = unchanged | 6-192 | |
| MOVB | loc16,AX.LSB | Store LSB of AX reg | 6-196 | |
| MOVB | loc16,AX.MSB | Store MSB of AX reg | 6-198 | |
| NEG | AX | Negate AX register | 6-245 | |
| NOT | AX | Complement AX register | 6-256 | |
| OR | AX,loc16 | Bitwise OR | 6-259 | |
| OR | loc16,AX | Bitwise OR | 6-263 | |
| ORB | AX,#8bit | Bitwise OR 8-bit value | 6-264 | |
| SUB | AX,loc16 | Subtract specified location from AX | 6-338 | |
| SUB | loc16,AX | Subtract AX from specified location | 6-339 | |
| SUBR | loc16,AX | Reverse-subtract specified location from AX | 6-354 | |
| SXTB | AX | Sign extend LSB of AX reg into MSB | | |
| XOR | AX,loc16 | Bitwise exclusive OR | 6-384 | |
| XORB | AX,#8bit | Bitwise exclusive OR 8-bit value | 6-387 | |
| XOR | loc16,AX | Bitwise exclusive OR | 6-385 | |
| 16-Bit ACC F | Register Operations | | | |
| ADD | ACC,loc16 {<< 016} | Add value to accumulator | 6-25 | |
| ADD | ACC,#16bit {<< 015} | Add value to accumulator | 6-22 | |
| ADD | ACC,loc16 << T | Add shifted value to accumulator | 6-24 | |
| ADDB | ACC,#8bit | Add 8-bit constant to accumulator | 6-30 | |
| ADDCU | ACC,loc16 | Add unsigned value plus carry to accumulator | 6-35 | |
| ADDU | ACC,loc16 | Add unsigned value to accumulator | 6-39 | |
| AND | ACC,loc16 | Bitwise AND | 6-44 | |
| AND | ACC,#16bit {<< 016} | Bitwise AND | 6-43 | |

Table 6-2. Register Operations (Continued)

| Mnemonic | | Description | Page |
|--------------|-----------------------------|---|-------|
| 16-Bit ACC F | Register Operations (Contin | ued) | |
| MOV | ACC,loc16 {<< 016} | Load accumulator with shift | 6-159 |
| MOV | ACC,#16bit {<< 015} | Load accumulator with shift | 6-159 |
| MOV | loc16,ACC << 18 | Save low word of shifted accumulator | 6-167 |
| MOV | ACC,loc16 << T | Load accumulator with shift | 6-158 |
| MOVB | ACC,#8bit | Load accumulator with 8-bit value | 6-187 |
| MOVH | loc16,ACC << 18 | Save high word of shifted accumulator | 6-202 |
| MOVU | ACC,loc16 | Load accumulator with unsigned word | 6-220 |
| SUB | ACC,loc16 << T | Subtract shifted value from accumulator | 6-335 |
| SUB | ACC,loc16 {<< 016} | Subtract shifted value from accumulator | 6-333 |
| SUB | ACC,#16bit {<< 015} | Subtract shifted value from accumulator | 6-337 |
| SUBB | ACC,#8bit | Subtract 8-bit value | 6-340 |
| SBBU | ACC,loc16 | Subtract unsigned value plus inverse borrow | 6-317 |
| SUBU | ACC,loc16 | Subtract unsigned 16-bit value | 6-356 |
| OR | ACC,loc16 | Bitwise OR | 6-257 |
| OR | ACC,#16bit {<< 016} | Bitwise OR | 6-258 |
| XOR | ACC,loc16 | Bitwise exclusive OR | 6-382 |
| XOR | ACC,#16bit {<< 016} | Bitwise exclusive OR | 6-383 |
| ZALR | ACC,loc16 | Zero AL and load AH with rounding | 6-394 |
| 32-Bit ACC F | legister Operations | | |
| ABS | ACC | Absolute value of accumulator | 6-19 |
| ABSTC | ACC | Absolute value of accumulator and load TC | 6-20 |
| ADDL | ACC,loc32 | Add 32-bit value to accumulator | 6-36 |
| ADDL | loc32,ACC | Add accumulator to specified location | 6-38 |
| ADDCL | ACC,loc32 | Add 32-bit value plus carry to accumulator | 6-34 |
| ADDUL | ACC,loc32 | Add 32-bit unsigned value to accumulator | 6-41 |
| ADDL | ACC,P << PM | Add shifted P to accumulator | 6-37 |

Table 6-2. Register Operations (Continued)

| Mnemonic | | Description | Page |
|--------------|-----------------------------|---|-------|
| 32-Bit ACC R | legister Operations (Contin | ued) | |
| ASRL | ACC,T | Arithmetic shift right of accumulator by T(4:0) | 6-57 |
| CMPL | ACC,loc32 | Compare 32-bit value | 6-80 |
| CMPL | ACC,P << PM | Compare 32-bit value | 6-81 |
| CSB | ACC | Count sign bits | 6-83 |
| LSL | ACC,116 | Logical shift left 1 to 16 places | 6-133 |
| LSL | ACC,T | Logical shift left by T(3:0) = 015 | 6-134 |
| LSRL | ACC,T | Logical shift right by T(4:0) | 6-144 |
| LSLL | ACC,T | Logical shift left by T(4:0) | 6-139 |
| MAXL | ACC,loc32 | Find the 32-bit maximum | 6-152 |
| MINL | ACC,loc32 | Find the 32-bit minimum | 6-155 |
| MOVL | ACC,loc32 | Load accumulator with 32 bits | 6-204 |
| MOVL | loc32,ACC | Store 32-bit accumulator | 6-206 |
| MOVL | P,ACC | Load P from the accumulator | 6-212 |
| MOVL | ACC,P << PM | Load the accumulator with shifted P | 6-205 |
| MOVL | loc32,ACC,COND | Store ACC conditionally | 6-207 |
| NORM | ACC,XARn++/ | Normalize ACC and modify selected auxiliary register. | 6-253 |
| NORM | ACC,*ind | C2XLP compatible Normalize ACC operation | 6-251 |
| NEG | ACC | Negate ACC | 6-244 |
| NEGTC | ACC | If TC is equivalent to 1, negate ACC | 6-248 |
| NOT | ACC | Complement ACC | 6-255 |
| ROL | ACC | Rotate ACC left | 6-310 |
| ROR | ACC | Rotate ACC right | 6-311 |
| SAT | ACC | Saturate ACC based on OVC value | 6-313 |
| SFR | ACC,116 | Shift accumulator right by 1 to 16 places | 6-325 |
| SFR | ACC,T | Shift accumulator right by $T(3:0) = 015$ | 6-326 |
| SUBBL | ACC,loc32 | Subtract 32-bit value plus inverse borrow | 6-343 |

Table 6-2. Register Operations (Continued)

6-9

| Mnemonic | | Description | Page |
|--------------|-------------------------------|--|-------|
| 32-Bit ACC R | legister Operations (Contin | ued) | |
| SUBCU | ACC,loc16 | Subtract conditional 16-bit value | 6-345 |
| SUBCUL | ACC,loc32 | Subtract conditional 32-bit value | 6-347 |
| SUBL | ACC,loc32 | Subtract 32-bit value | 6-350 |
| SUBL | loc32,ACC | Subtract 32-bit value | 6-353 |
| SUBL | ACC,P << PM | Subtract 32-bit value | 6-351 |
| SUBRL | loc32,ACC | Reverse-subtract specified location from ACC | 6-355 |
| SUBUL | ACC,loc32 | Subtract unsigned 32-bit value | 6-357 |
| TEST | ACC | Test for accumulator equal to zero | 6-362 |
| 64-Bit ACC:P | Register Operations | | |
| ASR64 | ACC:P,#116 | Arithmetic shift right of 64-bit value | 6-55 |
| ASR64 | ACC:P,T | Arithmetic shift right of 64-bit value by T(5:0) | 6-56 |
| CMP64 | ACC:P | Compare 64-bit value | 6-77 |
| LSL64 | ACC:P,116 | Logical shift left 1 to 16 places | 6-137 |
| LSL64 | ACC:P,T | 64-bit logical shift left by T(5:0) | 6-138 |
| LSR64 | ACC:P,#116 | 64-bit logical shift right by 1 to 16 places | 6-142 |
| LSR64 | ACC:P,T | 64-bit logical shift right by T(5:0) | 6-143 |
| NEG64 | ACC:P | Negate ACC:P | 6-246 |
| SAT64 | ACC:P | Saturate ACC:P based on OVC value | 6-314 |
| P or XT Regi | ster Operations (P, PH, PL, 2 | XT, T, TL) | |
| ADDUL | P,loc32 | Add 32-bit unsigned value to P | 6-40 |
| MAXCUL | P,loc32 | Conditionally find the unsigned maximum | 6-150 |
| MINCUL | P,loc32 | Conditionally find the unsigned minimum | 6-154 |
| MOV | PH,loc16 | Load the high half of the P register | 6-177 |
| MOV | PL,loc16 | Load the low half of the P register | 6-178 |
| MOV | loc16,P | Store lower half of shifted P register | 6-174 |
| MOV | T,loc16 | Load the upper half of the XT register | 6-180 |

Table 6–2. Register Operations (Continued)

| Mnemonic | | Description | Page |
|---------------|-------------------------------|--|-------|
| P or XT Regis | ster Operations (P, PH, PL, 2 | XT, T, TL) (Continued) | |
| MOV | loc16,T | Store the T register | 6-175 |
| MOV | TL,#0 | Clear the lower half of the XT register | 6-181 |
| MOVA | T,loc16 | Load the T register and add the previous product | 6-183 |
| MOVAD | T,loc16 | Load T register | 6-185 |
| MOVDL | XT,loc32 | Store XT and load new XT | 6-201 |
| MOVH | loc16,P | Save the high word of the P register | 6-203 |
| MOVL | P,loc32 | Load the P register | 6-213 |
| MOVL | loc32,P | Store the P register | 6-209 |
| MOVL | XT,loc32 | Load the XT register | 6-216 |
| MOVL | loc32,XT | Store the XT register | 6-211 |
| MOVP | T,loc16 | Load the T register and store P in the accumulator | 6-217 |
| MOVS | T,loc16 | Load T and subtract P from the accumulator | 6-218 |
| MOVX | TL,loc16 | Load lower half of XT with sign extension | 6-224 |
| SUBUL | P,loc32 | Subtract unsigned 32-bit value | 6-358 |
| 16x16 Multip | ly Operations | | |
| DMAC | ACC:P,loc32,*XAR7/++ | 16-bit dual multiply and accumulate | 6-86 |
| MAC | P,loc16,0:pma | Multiply and accumulate | 6-145 |
| MAC | P,loc16,*XAR7/++ | Multiply and Accumulate | 6-147 |
| MPY | P,T,loc16 | 16 X 16 multiply | 6-230 |
| MPY | P,loc16,#16bit | 16 X 16-bit multiply | 6-229 |
| MPY | ACC,T,loc16 | 16 X 16-bit multiply | 6-228 |
| MPY | ACC,loc16,#16bit | 16 X 16-bit multiply | 6-227 |
| MPYA | P,loc16,#16bit | 16 X 16-bit multiply and add previous product | 6-231 |
| MPYA | P,T,loc16 | 16 X 16-bit multiply and add previous product | 6-233 |
| MPYB | P,T,#8bit | Multiply signed value by unsigned 8-bit constant | 6-236 |
| MPYS | P,T,loc16 | 16 X 16-bit multiply and subtract | 6-237 |

Table 6-2. Register Operations (Continued)

| Mnemonic | | Description | Page |
|--------------|---------------------------|--|-------|
| 16x16 Multip | ly Operations (Continued) | | |
| MPYB | ACC,T,#8bit | Multiply by 8-bit constant | 6-235 |
| MPYU | ACC,T,loc16 | 16 X 16-bit unsigned multiply | 6-240 |
| MPYU | P,T,loc16 | Unsigned 16 X 16 multiply | 6-239 |
| MPYXU | P,T,loc16 | Multiply signed value by unsigned value | 6-242 |
| MPYXU | ACC,T,loc16 | Multiply signed value by unsigned value | 6-241 |
| SQRA | loc16 | Square value and add P to accumulator | 6-329 |
| SQRS | loc16 | Square value and subtract from accumulator | 6-331 |
| XMAC | P,loc16,*(pma) | C2xLP source-compatible multiply and accumulate | 6-378 |
| XMACD | P,loc16,*(pma) | C2xLP source-compatible multiply and accumulate with data move | 6-380 |
| 32x32 Multip | ly Operations | | |
| IMACL | P,loc32,*XAR7/++ | Signed 32 X 32-bit multiply and accumulate (lower half) | 6-100 |
| IMPYAL | P,XT,loc32 | Signed 32-bit multiply (lower half) and add previous P | 6-103 |
| IMPYL | P,XT,loc32 | Signed 32 X 32-bit multiply (lower half) | 6-106 |
| IMPYL | ACC,XT,loc32 | Signed 32 X 32-bit multiply (lower half) | 6-105 |
| IMPYSL | P,XT,loc32 | Signed 32-bit multiply (lower half) and subtract P | 6-107 |
| IMPYXUL | P,XT,loc32 | Signed 32 X unsigned 32-bit multiply (lower half) | 6-109 |
| QMACL | P,loc32,*XAR7/++ | Signed 32 X 32-bit multiply and accumulate (upper half) | 6-300 |
| QMPYAL | P,XT,loc32 | Signed 32-bit multiply (upper half) and add previous P | 6-302 |
| QMPYL | ACC,XT,loc32 | Signed 32 X 32-bit multiply (upper half) | 6-305 |
| QMPYL | P,XT,loc32 | Signed 32 X 32-bit multiply (upper half) | 6-304 |
| QMPYSL | P,XT,loc32 | Signed 32-bit multiply (upper half) and subtract pre- vious P | 6-306 |
| QMPYUL | P,XT,loc32 | Unsigned 32 X 32-bit multiply (upper half) | 6-308 |
| QMPYXUL | P,XT,loc32 | Signed 32 X unsigned 32-bit multiply (upper half) | 6-309 |
| Direct Memo | ry Operations | | |
| ADD | loc16,#16bitSigned | Add constant to specified location | 6-29 |

 Table 6–2. Register Operations (Continued)

| Mnemonic | | Description | Page |
|-------------|---------------------------|---------------------------------------|-------|
| Direct Memo | ry Operations (Continued) | | |
| AND | loc16,#16bitSigned | Bitwise AND | 6-50 |
| CMP | loc16,#16bitSigned | Compare | 6-75 |
| DEC | loc16 | Decrement by 1 | 6-84 |
| DMOV | loc16 | Data move contents of 16-bit location | 6-89 |
| INC | loc16 | Increment by 1 | 6-113 |
| MOV | *(0:16bit),loc16 | Move value | 6-156 |
| MOV | loc16,*(0:16bit) | Move value | 6-165 |
| MOV | loc16,#16bit | Save 16-bit constant | 6-164 |
| MOV | loc16,#0 | Clear 16-bit location | 6-166 |
| MOVB | loc16,#8bit,COND | Store byte conditionally | 6-194 |
| OR | loc16,#16bit | Bitwise OR | 6-262 |
| TBIT | loc16,#bit | Test bit | 6-359 |
| TBIT | loc16,T | Test bit specified by T register | 6-360 |
| TCLR | loc16,#bit | Test and clear specified bit | 6-361 |
| TSET | loc16,#bit | Test and set specified bit | 6-365 |
| XOR | loc16,#16bit | Bitwise exclusive OR | 6-386 |
| IO Space Op | erations | | |
| IN | loc16,*(PA) | Input data from port | 6-111 |
| OUT | *(PA),loc16 | Output data to port | 6-265 |
| UOUT | *(PA),loc16 | Unprotected output data to I/O port | 6-366 |
| Program Spa | ace Operations | | |
| PREAD | loc16,*XAR7 | Read from program memory | 6-282 |
| PWRITE | *XAR7,loc16 | Write to program memory | 6-299 |
| XPREAD | loc16,*AL | C2xLP source-compatible program read | 6-389 |
| XPREAD | loc16,*(pma) | C2xLP source-compatible program read | 6-388 |
| XPWRITE | *AL,loc16 | C2xLP source-compatible program write | 6-390 |

Table 6–2. Register Operations (Continued)

| Mnemonic | | Description | Page | |
|-------------------------------|-----------------------|---|-------|--|
| Branch/Call/Return Operations | | | | |
| В | 16bitOff,COND | Conditional branch | 6-58 | |
| BANZ | 16bitOff,ARn | Branch if auxiliary register not equal to zero | 6-59 | |
| BAR | 16bOf,ARn,ARn,EQ/NEQ | Branch on auxiliary register comparison | 6-60 | |
| BF | 16bitOff,COND | Branch fast | 6-61 | |
| FFC | XAR7,22bitAddr | Fast function call | 6-95 | |
| IRET | | Interrupt return | 6-116 | |
| LB | 22bitAddr | Long branch | 6-120 | |
| LB | *XAR7 | Long indirect branch | 6-119 | |
| LC | 22bitAddr | Long call immediate | 6-122 | |
| LC | *XAR7 | Long indirect call | 6-121 | |
| LCR | 22bitAddr | Long call using RPC | 6-123 | |
| LCR | *XARn | Long indirect call using RPC | 6-124 | |
| LOOPZ | loc16,#16bit | Loop while zero | 6-127 | |
| LOOPNZ | loc16,#16bit | Loop while not zero | 6-125 | |
| LRET | | Long return | 6-130 | |
| LRETE | | Long return and enable interrupts | 6-131 | |
| LRETR | | Long return using RPC | 6-132 | |
| RPT | #8bit/loc16 | Repeat next instruction | 6-312 | |
| SB | 8bitOff,COND | Short conditional branch | 6-316 | |
| SBF | 8bitOff,EQ/NEQ/TC/NTC | Short fast conditional branch | 6-318 | |
| XB | pma | C2XLP source-compatible branch | 6-369 | |
| XB | pma,COND | C2XLP source-compatible conditional branch | 6-370 | |
| XB | pma,*,ARPn | C2XLP source-compatible branch function call | 6-369 | |
| XB | *AL | C2XLP source-compatible function call | 6-368 | |
| XBANZ | pma,*ind{,ARPn} | C2XLP source-compatible branch if ARn is not zero | 6-372 | |
| XCALL | pma | C2XLP source-compatible call | 6-375 | |

Table 6–2. Register Operations (Continued)

| Mnemonic | | Description | Page | | |
|---------------|--|--|-------|--|--|
| Branch/Call/I | Branch/Call/Return Operations (Continued) | | | | |
| XCALL | pma,COND | C2XLP source-compatible conditional call | 6-376 | | |
| XCALL | pma,*,ARPn | C2XLP source-compatible call with ARP modification | 6-375 | | |
| XCALL | *AL | C2XLP source-compatible indirect call | 6-374 | | |
| XRET | | Alias for XRETC UNC | 6-391 | | |
| XRETC | COND | C2XLP source-compatible conditional return | 6-392 | | |
| Interrupt Reg | jister Operations | | | | |
| AND | IER,#16bit | Bitwise AND to disable specified CPU interrupts | 6-46 | | |
| AND | IFR,#16bit | Bitwise AND to clear pending CPU interrupts | 6-47 | | |
| IACK | #16bit | Interrupt acknowledge | 6-97 | | |
| INTR | INT1//INT14 NMI EMUINT DLOGINT RTOSINT | Emulate hardware interrupts | 6-114 | | |
| MOV | IER,loc16 | Load the interrupt-enable register | 6-163 | | |
| MOV | loc16,IER | Store interrupt enable register | 6-172 | | |
| OR | IER,#16bit | Bitwise OR | 6-260 | | |
| OR | IFR,#16bit | Bitwise OR | 6-261 | | |
| TRAP | #031 | Software trap | 6-363 | | |
| Status Regis | ter Operations (ST0, ST1) | | | | |
| CLRC | Mode | Clear status bits | 6-72 | | |
| CLRC | XF | Clear the XF status bit and output signal | 6-71 | | |
| CLRC | AMODE | Clear the AMODE bit | 6-67 | | |
| C28ADDR | | Clear the AMODE status bit | 6-64 | | |
| CLRC | OBJMODE | Clear the OBJMODE bit | 6-69 | | |
| C27OBJ | | Clear the OBJMODE bit | 6-63 | | |
| CLRC | M0M1MAP | Clear the M0M1MAP bit | 6-68 | | |
| C27MAP | | Set the M0M1MAP bit | 6-62 | | |

Table 6-2. Register Operations (Continued)

| Mnemonic | | Description | Page |
|--------------|-----------------------------|--|-------|
| Status Regis | ter Operations (ST0, ST1) (| Continued) | |
| CLRC | OVC | Clear OVC bits | 6-70 |
| ZAP | OVC | Clear overflow counter | 6-395 |
| DINT | | Disable maskable interrupts (set INTM bit) | 6-85 |
| EINT | | Enable maskable interrupt (clear INTM bit) | 6-92 |
| MOV | PM,AX | Load product shift mode bits $PM = AX(2:0)$ | 6-179 |
| MOV | OVC,loc16 | Load the overflow counter | 6-176 |
| MOVU | OVC,loc16 | Load overflow counter with unsigned value | 6-222 |
| MOV | loc16,OVC | Store the overflow counter | 6-173 |
| MOVU | loc16,OVC | Store the unsigned overflow counter | 6-221 |
| SETC | Mode | Set multiple status bits | 6-320 |
| SETC | XF | Set XF bit and output signal | 6-324 |
| SETC | M0M1MAP | Set M0M1MAP bit | 6-65 |
| C28MAP | | Set the M0M1MAP bit | 6-322 |
| SETC | OBJMODE | Set OBJMODE bit | 6-66 |
| C28OBJ | | Set the OBJMODE bit | 6-323 |
| SETC | AMODE | Set AMODE bit | |
| LPADDR | | Alias for SETC AMODE | 6-129 |
| SPM | PM | Set product shift mode bits | 6-327 |
| Miscellaneou | is Operations | | |
| ABORTI | | Abort interrupt | 6-18 |
| ASP | | Align stack pointer | 6-52 |
| EALLOW | | Enable access to protected space | 6-90 |
| IDLE | | Put processor in IDLE mode | 6-98 |
| NASP | | Un-align stack pointer | 6-243 |
| NOP | {*ind} | No operation with optional indirect address modification | 6-250 |
| ZAPA | | Zero accumulator P register and OVC | 6-396 |
| EDIS | | Disable access to protected space | 6-91 |

Table 6–2. Register Operations (Continued)

6-16

Table 6–2. Register Operations (Continued)

| Mnemonic | Description | Page |
|--------------------------------------|------------------|------|
| Miscellaneous Operations (Continued) | | |
| ESTOP0 | Emulation Stop 0 | 6-93 |
| ESTOP1 | Emulation Stop 1 | 6-94 |

| ABORTI | | | | A | bort Int | terrupt | | |
|--|----------|--|--|---|---------------------------------|---------------------------------------|--|--|
| | SYNTAX C | PTIONS | OPCODE | OBJMODE RPT | | CYC | | |
| ABORTI | | | 0000 0000 0000 0001 | Х | - | 2 | | |
| Operands | | None | | | | | | |
| Description | | Abort interrupt. This instruction is available for emulation purposes. Generally, a program uses the IRET instruction to return from an interrupt. The IRET instruction restores all of the values that were saved to the stack during the automatic context save. In restoring status register ST1 and the debug status register (DBGSTAT), IRET restores the debug context that was present before the interrupt. | | | | | | |
| | | In some target applications, you might have interrupts that must not be returned from by the IRET instruction. Not using IRET can cause a problem for the emulation logic, because the emulation logic assumes that the original debug context will be restored. The abort interrupt (ABORTI) instruction is provided as a means to indicate that the debug context will not be restored and the debug logic needs to be reset to its default state. As part of its operation, the ABORTI instruction: | | | | | | |
| | | Sets the DBGN | /l bit in ST1. This disables d | lebug events. | | | | |
| | | Modifies select the debug con interrupt occurr | t bits in the DBGSTAT regis itext. If the CPU was in th red, the CPU does not halt | ter. This effec e debug-halt when the inter | t is a re state b rupt is | esetting of before the aborted. | | |
| | | The ABORTI instruction does not modify the DBGIER, the IER, the INTM bit or any analysis registers (for example, registers used for breakpoints, watch points, and data logging). | | | | | | |
| Flags and Modes | DBGM | The DBGM bit is se | ət. | | | | | |
| Repeat This instruction it resets the rep | | | not repeatable. If this instruction follows the RPT instruction, at counter (RPTC) and executes only once. | | | | | |

6-18

ABS ACC

Absolute Value of Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ABS ACC | 1111 1111 0101 0110 | Х | - | 1 |

| ACC | Accumulator register | | | | | | |
|--------|---|--|--|--|--|--|--|
| | The content of the ACC re | egister is replaced with its absolute value: | | | | | |
| | <pre>if(ACC = 0x8000 0000) V = 1; If (OVM = 1) ACC = 0x7FFF FFFF; else ACC = 0x8000 0000; else if(ACC < 0) ACC = -ACC;</pre> | | | | | | |
| Ν | After the operation, the N f | ag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | |
| Z | After the operation, the Z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. | | | | | |
| С | C is cleared by this opera | C is cleared by this operation. | | | | | |
| V | If (ACC = $0x8000\ 0000$) at the start of the operation, this is considered an overflow value and V is set. Otherwise, V is not affected. | | | | | | |
| OVM | If (ACC = 0x8000 0000) a overflow value, and the A0 of OVM: If OVM is cleared ACC will be saturated to 0 | t the start of the operation, this is considered an CC value after the operation depends on the state ACC will be filled with 0x8000 0000. If OVM is set 0x7FFF FFFF. | | | | | |
| | This instruction is not r instruction, it resets the re | epeatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. | | | | | |
| ; Take | absolute value of VarA | , make sure value is saturated: | | | | | |
| MOVL | ACC,@VarA | ; Load ACC with contents of VarA | | | | | |
| ABS | ACC | : Absolute of ACC and saturate | | | | | |
| MOVL | @VarA,ACC | ; Store result into VarA | | | | | |
| | ACC N Z C V OVM ; Take MOVL SETC ABS MOVL | ACCAccumulator registerThe content of the ACC registerif (ACC = $0 \times 8000\ 0000$)V = 1;If (OVM = 1)ACC = $0 \times 7FFF$ FFFF;elseACC = $0 \times 8000\ 0000$;elseif (ACC < 0)ACC = $-ACC$;NAfter the operation, the N flZAfter the operation, the ZCC is cleared by this operationVIf (ACC = $0 \times 8000\ 0000$) a overflow value and V is setOVMIf (ACC = $0 \times 8000\ 0000$) a overflow value, and the AC of OVM: If OVM is cleared, ACC will be saturated to CThis instruction is not r instruction, it resets the ref; Take absolute value of VarA MOVL ACC, @VarA SETCMOVL@VarA, ACC | | | | | |

ABSTC ACC

Absolute Value of Accumulator and Load TC

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ABSTC ACC | 0101 0110 0101 1111 | 1 | - | 1 |

| Operands | ACC | Accumulator register | | | |
|-------------|-----|--|--|--|--|
| Description | | Replace the content of the ACC register with its absolute value and load the test control (TC) bit with the sign bit XORed with the previous value of the test control bit: | | | |
| | | <pre>if(ACC = 0x8000 0000) { If (OVM = 1) ACC = 0x7FFF FFFF; else ACC = 0x8000 0000; V = 1; TC = TC XOR 1; { else { if(ACC < 0) ACC = -ACC; TC = TC XOR 1; } C = 0;</pre> | | | |
| Flags and | N | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | |
| Modes | Z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. | | | |
| | С | The C flag bit is cleared. | | | |
| | V | If (ACC = $0x8000\ 0000$) at the start of the operation, this is considered an overflow value and V is set; otherwise, V is not affected. | | | |
| | тс | If (ACC < 0) at the start of the operation, then $TC = TC XOR 1$; otherwise, TC is not affected. | | | |
| | OVM | If at the start of the operation, ACC = $0x8000\ 0000$, then this is considered an overflow value and the ACC value after the operation depends on OVM. If OVM is cleared and TC == 1, ACC will be filled with 0x8000\ 0000. If OVM is set and TC = 1, ACC will be saturated to 0x7FFF FFFF. | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | |

```
Example
           ; Calculate signed: Quot16 = Num16/Den16, Rem16 = Num16%Den16
            CLRC TC
                                      ; Clear TC flag, used as sign flag
            MOV ACC,@Den16 << 16
                                     ; AH = Den16, AL = 0
            ABSTC ACC
                                      ; Take abs value, TC = sign ^ TC
            MOV T,@AH
                                     ; Temp save Den16 in T register
            MOV ACC,@Num16 << 16
                                     ; AH = Num16, AL = 0
            ABSTC ACC
                                      ; Take abs value, TC = sign ^ TC
            MOVU ACC,@AH
                                     ; AH = 0, AL = Num16
            RPT
                 #15
                                     ; Repeat operation 16 times
           SUBCU ACC,@T
                                     ; Conditional subtract with Den16
            MOV @Rem16,AH
                                     ; Store remainder in Rem16
            MOV ACC,@AL << 16
                                  ; AH = Quot16, AL = 0
            NEGTC ACC
                                      ; Negate if TC = 1
            MOV @Quot16,AH
                                     ; Store quotient in Quot16
```

ADD ACC,#16bit<<#0..15

Add Value to Accumulator

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|----------------------|--|--|----------------------------------|---------|----------------|--|--|--|
| ADD ACC,#16 | oit<<#015 | | 1111 1111 0001 SHFT CCCC CCCC CCCC CCCC | Х | - | 1 | | | |
| Operands | ACC | Accumulator register | | | | | | | |
| • | #16bit | 16-bit immediate constant value | | | | | | | |
| | #015 | Shift value (default | is "<< #0" if no value spec | ified) | | | | | |
| Description | | Add the left shifted 16-bit immediate constant value to the ACC register. The shifted value is sign extended if sign extension mode is turned on (SXM = 1) else the shifted value is zero extended (SXM = 0). The lower bits of the shifted value are zero filled: if (SXM = 1) // sign extension mode enabled ACC = ACC + S:16bit << shift value; else // sign extension mode disabled ACC = ACC + 0:16bit << shift value; Smart Encoding: If #16bit is an 8-bit number and the shift is 0, then the assembler will encode this instruction as ADDB ACC, #8bit to improve efficiency. To override this encoding, use the ADDW ACC, #16bit instruction alias. | | | | | | | |
| Flags and Modes | z | After the addition, the Z flag is set if the ACC value is zero, else the flag is cleared. | | | | | | | |
| | Ν | ne N flag is set if bit 31 of tl | the ACC is 1, else the flag is | | | | | | |
| | С | If the addition gene | rates a carry, C is set; othe | rwise C is cleared. | | | | | |
| | V | If an overflow occurs, V is set; otherwise V is not affected. | | | | | | | |
| | OVC SXM OVM | If (OVM = 0, disabled) then if the operation generates a positive overflow then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. If sign extension mode bit is set; then the 16-bit immediate constant will b sign-extended before the addition. Else, the value will be zero extended. If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFFFFF) or maximum negative (0x8000000) if the operation overflowed. | | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ir s the repeat counter (RPTC | struction follo) and execute | ows the | e RPT once. | | | |
| Example | ; Calcul SETC SXN | late signed value 1 | : ACC = (VarB << 10) + ; Turn sign extension | (23 << 6); on mode on | | | | | |

| MOV | ACC,@VarB << #10 | ; | Load AC | CC wit | h VarB | left | shifted | by | 10 |
|-----|------------------|---|---------|--------|---------|--------|----------|----|----|
| ADD | ACC,#23 << #6 | ; | Add 23 | left | shifted | l by 6 | 5 to ACC | | |
ADD ACC,loc16 << T

Add Value to Accumulator

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|--|--|---|-------------------------------|--------------------------------|--|
| ADD ACC,loc1 | ô<< T | | 0101 0110 0010 0011 0000 0000 LLLL LLLL | 1 | Y | N+1 | |
| Operands | ACC loc16 T | Accumulator registe Addressing mode (s Upper 16 bits of the | Accumulator register Addressing mode (see Chapter 5) Upper 16 bits of the multiplicand register, XT(31:16) | | | | |
| Description | | Add to the ACC regis to by the "loc16" add least significant bits of bits of T are ignored mode is turned on (SXM = 0). The lower if $(SXM = 1)$ ACC = ACC + else ACC = ACC + | Add to the ACC register the left-shifted contents of the 16-bit location point oby the "loc16" addressing mode. The shift value is specified by the least significant bits of the T register, T(3:0) = shift value = 015. Higher of bits of T are ignored. The shifted value is sign extended if sign extended is surred on (SXM = 1) else the shifted value is zero extended (SXM = 0). The lower bits of the shifted value are zero filled: if (SXM = 1) // sign extension mode enabled ACC = ACC + S: [loc16] << T(3:0); else // sign extension mode disabled ACC = ACC + 0: [loc16] << T(3:0); | | | | |
| Flags and Modes | Z N C | After the addition, the After the addition, the If the addition genera | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. If the addition generates a carry, C is set; otherwise C is cleared. | | | | |
| | V | If an overflow occurs | s, V is set; otherwise V is no | ot affected. | | | |
| | ovc | If OVM = 0, disabled counter is incremented the counter is decre affected by the opera | If $OVM = 0$, disabled and the operation generates a positive overflow, then counter is incremented; if the operation generates a negative overflow, the the counter is decremented. If $OVM = 1$, enabled, then the counter is affected by the operation. | | | | |
| | SXM | "loc16" field, will be s zero extended. | ign extended before the add | operand, add dition. Else, the | ressed e value | by the will be | |
| | OVM | If overflow mode bit is (0x7FFFFFFF) or poverflowed. | s set; then the ACC value wi maximum negative (0x80 | ll saturate max 000000) if t | timum p he ope | ositive eration | |
| Repeat | | If this operation is re The state of the Z, N an intermediate ove overflows, if overflow | peated, then the instruction , C flags will reflect the final r erflow occurs. The OVC f w mode is disabled. | will be execut result. The V fla lag will count | ed N+1 ag will b interm | times. be set if bediate | |
| Example | ; Cal SETC MOV MOV MOV ADD | Iculate signed value SXM T,@SA ACC,@VarA << T T,@SB ACC,@VarB << T | : ACC = (VarA << SB) + ; Turn sign extension ; Load T with shift ; Load in ACC shifted ; Load T with shift ; Add to ACC shifted | (VarB << Sl on mode on value in SZ ed contents value in SE d contents of | B) of Van of Vari | rA | |

ADD ACC,loc16 << #0..16

Add Value to Accumulator

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--|----------------------|--|---|---|---------------------|---|
| ADD ACC,loc1 | 6<<#0 | | 1000 0001 LLLL LLLL | 1 | Y | N+1 |
| ADD ACC,loc1 | 6 << #11 | 5 | 0101 0110 0000 0100 0000 SHFT LLLL LLLL | 1 | Y | N+1 |
| ADD ACC,loc1 | 6 << #16 | | 0000 0101 LLLL LLLL | Х | Y | N+1 |
| ADD ACC,loc1 | 6<<015 | | 1010 SHFT LLLL LLLL | 0 | - | N+1 |
| Operands | ACC loc16 #016 | Accumulator registe Addressing mode (s Shift value (default i | Accumulator register Addressing mode (see Chapter 5) Shift value (default is "<< #0" if no value specified) | | | |
| Description | | Add the left shifted 16-bit location pointed to by the "loc16" addressing mode to the ACC register. The shifted value is sign extended if sign extension mode is turned on (SXM = 1) else the shifted value is zero extended (SXM = 0). The lower bits of the shifted value are zero filled: if (SXM = 1) // sign extension mode enabled ACC = ACC + S: [loc16] << shift value; else // sign extension mode disabled ACC = ACC + 0: [loc16] << shift value: | | | | y mode ension tended |
| Flags and Modes | Z N C V | After the addition, the Z flag is set if ACC is zero, else Z is cleared. After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cl If the addition generates a carry, C is set; otherwise C is cleared. <i>Exception:</i> If a shift of 16 is used, the ADD instruction can set C but not c If an overflow occurs. V is set: otherwise V is not affected | | | leared. clear C. | |
| | ovc sxm | If (OVM = 0, disabled) then if the operation generates a positive overflow, the counter is incremented and if the operation generates a neg- overflow, then the counter is decremented. If (OVM = 1, enabled) ther counter is not affected by the operation. If sign extension mode bit is set; then the 16-bit operand, addressed by "loc16" field, will be sign extended before the addition. Else, the value w zero extended. If overflow mode bit is set; then the ACC value will saturate maximum pos (0x7FFFFFF) or maximum negative (0x8000000) if the opera- overflowed. | | | | w, then egative en the by the will be |
| | OVM | | | | | oositive eration |
| Repeat If the operation is repeatable, then the instruction will be executive times. The state of the Z, N, C flags will reflect the final result. The N be set if an intermediate overflow occurs. The OVC flag wintermediate overflows, if overflow mode is disabled. If the operative repeatable, the instruction will execute only once. | | | execute The V f ag will peratior | ed N+1 lag will count n is not | | |

| Example | ; Calculate signed value: | : ACC = VarA << 10 + VarB << 6; |
|---------|---------------------------|---|
| | SETC SXM | ; Turn sign extension mode on |
| | MOV ACC,@VarA << #10 | ; Load ACC with VarA left shifted by 10 |
| | ADD ACC,@VarB << #6 | ; Add VarB left shifted by 6 to ACC |

ADD AX, loc16

Add Value to AX

| | SYNTA | K OPTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-------------|--|--|--|------------------------------|--------------------------------|--|
| ADD AX, loc1 | 6 | | 1001 010A LLLL LLLL | Х | - | 1 | |
| Operands | AX loc16 | Accumulator high (AF Addressing mode (se | Accumulator high (AH) or accumulator low (AL) register Addressing mode (see Chapter 5) | | | | |
| Description | | Add the contents of the location pointed to by the "loc16" addressing the specified AX register (AH or AL) and store the result in the AX r AX = AX + [loc16]; | | | | node to gister: | |
| Flags and Modes | Ν | After the addition, AX the negative flag bit is | After the addition, AX is tested for a negative condition. If bit 15 of AX is 1, th the negative flag bit is set, otherwise it is cleared. | | | | |
| | z | After the addition, AX is tested for a zero condition. The zero flag operation results in $AX = 0$; otherwise it is cleared. | | | | | |
| | С | If the addition generat | tes a carry, C is set; otherw | ise, C is clear | ed. | | |
| | v | If an overflow occurs, overflow occurs if the positive direction. Sig max negative value (0 | , V is set; otherwise V is no result crosses the max pos ned negative overflow occu 0x8000) in the negative dire | ot affected. Si sitive value (0 urs if the resu ection. | igned p x7FFF lt cross | ositive) in the ses the | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | ∋ RPT ıce. | | |
| Example | ; Ad | d the contents of Va | rA with VarB and store | in VarC | | | |
| - | MOV | AL,@VarA | ; Load AL with conte | ents of Vari | Ą | | |
| | ADD | AL,@VarB | ; Add to AL contents | s of VarB | | | |
| | MOV | @VarC,AL | ; Store result in Va | arC | | | |

ADD loc16, AX

Add AX to Specified Location

| 5 | SYNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|---|---|-----------------------------------|----------|--------------------------------|--|--|
| ADD loc16, AX | | | 0111 001A LLLL LLLL | Х | - | 1 | | |
| Operands | loc16 AX | Addressing mode (se Accumulator high (Al | ddressing mode (see Chapter 5) ccumulator high (AH) or accumulator low (AL) register | | | | | |
| Description | | Add the contents of the specified AX register (AH or AL) to the location pointed to by the "loc16" addressing mode and store the results in location pointed to by "loc16": | | | | ocation ocation | | |
| | | [loc16] = [loc16] | [loc16] = [loc16] + AX; | | | | | |
| | | This is a read-modify | y-write operation. | | | | | |
| Flags and Modes | N | After the addition, [loc16] is tested for a negative condition. If bit 15 of [loc16] 1, then the negative flag bit is set, otherwise it is cleared. | | | | | | |
| | After the addition, [loc16] is tested for a zero condition. The zero flag bit the operation generates [loc16] = 0; otherwise it is cleared | | | | flag bit | is set if | | |
| | С | If the addition genera | tes a carry, C is set; otherw | /ise C is clear | ed. | | | |
| | V | If an overflow occurs, V is set; otherwise V is not affected. Signed positive overflow occurs if the result crosses the max positive value (0x7FFF) in the positive direction. Signed negative overflow occurs if the result crosses the max negative value (0x8000) in the negative direction. | | | | ositive) in the ses the | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Add MOV 2 ADD @ ; Add MOV 2 ADD @ | the contents of Va AL,@VarA @AR0,AL the contents of Va AH,@VarB @VarC,AH | rA to index register A ; Load AL with cont ; AR0 = AR0 + AL rB to VarC: ; Load AH with cont ; VarC = VarC + AH | R0: tents of Va tents of Va | rA rB | | | |

ADD loc16,#16bitSigned

Add Constant to Specified Location

| | SYNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|---------------|---|--|--|--|-----------------------------|----------------------------|--|--|--|
| ADD loc16,#16 | bitSigned | | 0000 1000 LLLL LLLL CCCC CCCC CCCC CCCC | Х | - | 1 | | | |
| Operands | loc16 | Addressing mode | (see Chapter 5) | | | | | | |
| | #16bit- Signed | 16-bit immediate | signed constant value | | | | | | |
| Description | | Add the specified signed 16-bit immediate constant to the sigr content of the location pointed to by the "loc16" addressing r store the 16-bit result in the location pointed to by "loc16": | | | | l 16-bit de and | | | |
| | | [loc16] = [loc: | 16] + 16bitSigned; | | | | | | |
| | | Smart Encoding: If loc16 = AL or assembler will er improve efficienc #16bitSigned inst | AH and #16bitSigned is neode this instruction as A y. To override this encodin ruction alias. | an 8-bit num \DDB AX, #1 ng, use the / | iber the 6bitSig ADDW | en the ned to loc16, | | | |
| Flags and | N | After the addition, i | f bit 15 of [loc16] is 1, then th | e N bit is set; e | else N c | leared. | | | |
| Modes | z | After the addition, | After the addition, if [loc16] is zero, the Z is set, else Z is cleared. | | | | | | |
| | С | If the addition gen | If the addition generates a carry, C is set; otherwise, C is cleared. | | | | | | |
| | v | If an overflow occu | urs, V is set; otherwise, V is | cleared. | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Calcu ; VarA = ; VarB = ADD @ ADD @ | .ate: : VarA + 10 : VarB - 3 VarA,#10 ; VarA = VarA + 10 VarB,#-3 ; VarB = VarB - 3 | | | | | | | |

ADDB ACC,#8bit

Add 8-bit Constant to Accumulator

| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | | |
|-------------|---|---|---|-------------|-----|-----|--|--|--|--|
| ADDB ACC,#8 | Bbit | | 0000 1001 CCCC CCCC | Х | - | 1 | | | | |
| Operands | ACC | Accumulator register | Accumulator register | | | | | | | |
| | #8bit | 8-bit immediate unsigned constant value | | | | | | | | |
| Description | | Add an 8-bit, zero-ex | tended constant to the ACC | C register: | | | | | | |
| | | ACC = ACC + 0:8bit; | | | | | | | | |
| Flags and | z | After the addition, the | After the addition, the Z flag is set if ACC is zero, else Z is cleared. | | | | | | | |
| Modes | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | | | |
| | C If the addition generates a carry, C is set; otherwise C is cleared. | | | | | | | | | |
| | v | If an overflow occurs, V is set; otherwise V is not affected. | | | | | | | | |
| | OVC | If (OVM = 0, disabled) the counter is increme then the counter is dec affected by the operat | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. | | | | | | | |
| | OVM | If overflow mode bit is set; then the ACC value will saturate maximum positiv (0x7FFFFFF) or maximum negative (0x80000000) if the operation overflowed. | | | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruc- tion, it resets the repeat counter (RPTC) and executes only once. | | | | | | | | |
| Example | ; Inc: MOVL ; ADDB ; MOVL (| rement contents of 32-bit location VarA: ACC,@VarA ; Load ACC with contents of VarA ACC,#1 ; Add 1 to ACC aVarA,ACC ; Store result back into VarA | | | | | | | | |

ADDB AX, #8bitSigned

Add 8-bit Constant to AX

| | SYNTAX OP | TIONS | | OP | CODE | OBJMODE | RPT | CYC |
|---|---|---|--|--|---|--|--------------------------------|---------------------------------|
| ADDB AX, #8 | bitSigned | | 1001 | 110A | CCCC CCCC | Х | - | 1 |
| Operands | AX #8bit- Signed | Accumulator high (AH) or accumulator low (AL) register 8-bit immediate signed 2s complement constant value (-128 to 127) | | | | | | |
| Description | | Add the sign extended 8-bit constant to the specified AX register (A or AL) and store the result in the AX register: AX = AX + S:8bit; | | | | (AH | | |
| Flags and Modes | Ν | After the addition, AX is tested for a negative condition. If bit 15 of AX is then the negative flag bit is set; otherwise it is cleared. | | | | AX is 1, | | |
| Z After the addition, AX is tested for a zero condition the operation results in AX = 0, otherwise it is | | | sted for a zero condition. The zero flag bit is set if $X = 0$, otherwise it is cleared | | | | | |
| | С | If the addition gen | erates | a cari | y, C is set; ot | nerwise C is cl | eared. | |
| | v | If an overflow occu overflow occurs if t positive direction. S max negative valu | urs, V i the res Signed ie (0x8 | s set; o sult cro l negat 3000) i | otherwise V is sses the max ive overflow o n the negative | not affected. S positive value (ccurs if the rest direction. | igned p 0x7FFF ult cross | oositive) in the ses the |
| Repeat | | This instruction is instruction, it rese once. | not re ets the | epeata repea | ble. If this inst t counter (RP | ruction follows TC) and execu | the RF | РТ У |
| Example | ; Add 2 MOV AL, ADDB AL MOV @Va MOV AL, ADDB AL MOV @Va | to VarA and subt @VarA ,#2 .rA,AL @VarB .,#-3 .rB,AL | ract ; ; ; ; ; | 3 from Load Add 1 Store Load Add 1 Store | n VarB: AL with con to AL the va e result in AL with con to AL the va e result in | ntents of Va Alue 0x0002 VarA Atents of Va Alue 0xFFFD VarB | rA (2) rB (-3) | |

ADDB SP, #7bit

Add 7-bit Constant to Stack Pointer

| SYNTAX OPTIONS | | | | OPCODE OBJMODE | | | | CYC |
|--------------------|------------|-------------------|--|----------------|--|------------------------|----------|----------|
| ADDB SP, #7b | bit | | | 11 | 11 1110 OCCC CCCC | Х | _ | 1 |
| Operands | SP | Stack | pointer | | | | | |
| | #7bit | 7-bit i | mmediate unsig | gne | d constant value | | | |
| Description | | Add a | Add a 7-bit unsigned constant to SP and store the result in SP: SP = SP + 0:7bit; | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This i it rese | nstruction is not ets the repeat co | rep oun | eatable. If this instruction ter (RPTC) and execute | follows the R | PT insti | ruction, |
| Example | FuncA: | | | ; | Function with local | variables | on sta | ack. |
| | ADI | OB SP, | #N | ; ; | Reserve N 16-bit wo local variables on | ords of spac stack: | e for | |
| | | | | | | | | |
| | SUE LRE | B SP, Tr | #N | ; ; | Deallocate reserved Return from functio | l stack spac on. | e. | |

ADDB XARn, #7bit

Add 7-bit Constant to Auxiliary Register

| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------------|--|--|--------------------------------|----------|----------|
| ADDB XARn, # | #7bit | | 1101 1nnn OCCC CCCC | Х | - | 1 |
| Operands | XARn | XAR0-XAR7, 32-bit a | uxiliary registers | | | |
| Description | | Add a 7-bit unsigned constant to XARn and store the result in XARn: XARn = XARn + 0:7bit; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is not r it resets the repeat co | repeatable. If this instruction unter (RPTC) and executes | follows the Rl s only once. | PT insti | ruction, |
| Example | MOVL | XAR1,#VarA | ; Initialize XAR1 p ; of VarA | oointer with | ı addr | ess |
| | MOVL ADDB | XAR2,*XAR1 XAR2,#10h | ; Load XAR2 with co ; XAR2 = VarA + 0x1 | ontents of N LO | /arA | |

ADDCL ACC, loc32

Add 32-bit Value Plus Carry to Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|--|---------|-----|-----|
| ADDCL ACC, loc32 | 0101 0110 0100 0000 xxxx xxxx LLLL LLLL | 1 | - | 1 |

| Operands | ACC | Accumulator registe | er |
|-------------|-------|--|---|
| | loc32 | Addressing mode (| see Chapter 5) |
| Description | | Add to the ACC reg "loc32" addressing r | ister the 32-bit content of the location pointed to by the node: |
| | | ACC = ACC + [loc | 32] + C; |
| Flags and | z | After the addition, t | he Z flag is set if the ACC is zero, else Z is cleared. |
| Modes | Ν | After the addition, the | he N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | С | The state of the car addition generates | rry bit before execution is included in the addition. If the a carry, C is set; otherwise C is cleared. |
| | v | If an overflow occu | rs, V is set; otherwise V is not affected. |
| | OVC | If (OVM = 0, disabl then the counter is overflow, then the c counter is not affec | ed) then if the operation generates a positive overflow, incremented and if the operation generates a negative counter is decremented. If (OVM = 1, enabled) then the ted by the operation. |
| | ΟνΜ | If overflow mode to positive (0x7FFFF) overflows. | bit is set; then the ACC value will saturate maximum FF) or maximum negative (0x80000000) if the operation |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this instruction follows the RPT the repeat counter (RPTC) and executes only once. |
| Example | ; Add | two 64-bit values | (VarA and VarB) and store result in VarC: |
| - | MOVL | ACC,@VarA+0 | ; Load ACC with contents of the low ; 32 bits of VarA |
| | ADDUL | ACC,@VarB+0 | ; Add to ACC the contents of the low ; 32 bits of VarB |
| | MOVL | <pre>@VarC+0,ACC</pre> | ; Store low 32-bit result into VarC |
| | MOVL | ACC,@VarA+2 | ; Load ACC with contents of the high ; 32 bits of VarA |
| | ADDCL | ACC,@VarB+2 | ; Add to ACC the contents of the high ; 32 bits of VarB with carry |
| | MOVL | <pre>@VarC+2,ACC</pre> | ; Store high 32-bit result into VarC |

ADDCU ACC, loc16

Add Unsigned Value Plus Carry to Accumulator

| | SYNTAX OF | TIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|-------------|--|---|---|---|------------------|-----------------------------|--|--|--|
| ADDCU ACC,I | oc16 | | 0000 1100 LLLL LLLL | Х | - | 1 | | | |
| Operands | ACC | Accumulator regis | ter | | | | | | |
| | 10010 | Addressing mode | (see Ghapter 5) | | | | | | |
| Description | | Add the 16-bit cont mode, zero extend register: | Add the 16-bit contents of the location pointed to by the "loc16" addressing mode, zero extended, plus the content of the carry flag bit to the ACC register: | | | | | | |
| | | ACC = ACC + 0: | loc16] + C; | | | | | | |
| | | | | | | | | | |
| Flags and | Z | After the addition, the | ne Z flag is set if the ACC va | lue is zero, els | e Z is c | leared. | | | |
| Modes | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | | |
| | С | The state of the car addition generates | The state of the carry bit before execution is included in the addition. If the addition generates a carry, C is set; otherwise C is cleared. | | | | | | |
| | V | If an overflow occu | If an overflow occurs, V is set; otherwise V is not affected. | | | | | | |
| | ovc | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. | | | | | | | |
| | OVM | If overflow mode to positive (0x7FFFF operation overflowe | bit is set; then the ACC va FFF) or maximum nega ed. | lue will satur tive (0x8000 | ate ma 00000) | ximum if the | | | |
| Repeat | | This instruction is instruction, it reset | not repeatable. If this ir this ir the repeat counter (RPTC | struction follo C) and execut | ows the | e RPT [,] once. | | | |
| Example | ; Add th MOVU ACC ADD ACC ADDU ACC ADD ACC ADD ACC ADD ACC | hree 32-bit unsig C,@VarAlow C,@VarAhigh << 16 C,@VarBlow C,@VarBhigh << 16 CC,@VarClow C,@VarChigh << 16 | ned variables by 16-bi ; AH = 0, AL = Van ; AH = VarAhigh, A ; ACC = ACC + 0:Va ; ACC = ACC + VarB ; ACC = ACC + VarC ; ACC = ACC + VarC | t parts: CAlow AL = VarAlow ArBlow Bhigh << 16 Clow + Carry Chigh << 16 | 7 7 | | | | |

ADDL ACC, loc32

Add 32-bit Value to Accumulator

| | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|---------------|--|---|--|-----------------------------------|--------------------|------------------|--|--|--|
| ADDL ACC, loc | :32 | | 0000 0111 LLLL LLLL | Х | Y | N+1 | | | |
| Operands | ACC | Accumulator registe | r See Chanter 5) | | | | | | |
| | 10032 | Addressing mode (s | ee Chapter 5) | | | | | | |
| Description | | Add to the ACC regis "loc32" addressing m | Add to the ACC register the 32-bit content of the location pointed to by the 'loc32' addressing mode: | | | | | | |
| | | ACC = ACC + [loc3] | 2]; | | | | | | |
| Flags and | N | After the addition, th | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| Modes | Z | After the addition, the Z flag is set if the ACC is zero, else Z is cleared. | | | | | | | |
| | С | If the addition gener | f the addition generates a carry, C is set; otherwise C is cleared. | | | | | | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | | | | |
| | OVC | If (OVM = 0, disable then the counter is i overflow, then the co counter is not affect | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. | | | | | | |
| | OVM | If overflow mode bi positive (0x7FFFFI overflows. | it is set; then the ACC va F) or maximum negative (0 | lue will satura 0x80000000) it | ate ma f the op | ximum eration | | | |
| Repeat | | If this operation is re The state of the Z, N an intermediate ove overflows, if overflow | If this operation is repeated, then the instruction will be executed N+1 times. The state of the Z, N, C flags will reflect the final result. The V flag will be set if an intermediate overflow occurs. The OVC flag will count intermediate overflows, if overflow mode is disabled. | | | | | | |
| Example | ; Calculate the 32-bit value: VarC = VarA + VarB MOVL ACC,@VarA ; Load ACC with contents of VarA ADDL ACC,@VarB ; Add to ACC the contents of VarB MOVL @VarC,ACC ; Store result into VarC | | | | | | | | |

ADDL ACC,P << PM

Add Shifted P to Accumulator

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | | |
|-----------------|------------------|---|---|---|--|---|--|
| ADDL ACC,P | << PM | | 0001 0000 1010 1100 | Х | Y | N+1 | |
| Note: This inst | ruction is an | alias for the "MOVA T,loc1 | 6" operation with "loc16 = @T" add | dressing mode. | - | | |
| Operands | ACC | Accumulator registe | er | | | | |
| • | Р | Product register | | | | | |
| | ~~ PM | Product shift mode | | | | | |
| | | | | | | | |
| Description | | Add to the ACC regis the product shift mod | ster the contents of the P rec de (PM): | gister, shifted a | as speci | ified by | |
| | | ACC = ACC + P << | PM | | | | |
| Flags and | z | After the addition, th | e Z flag is set if the ACC val | ue is zero, els | e Z is c | leared. | |
| Modes | Ν | After the addition, th | e N flag is set if bit 31 of the | ACC is 1, els | e N is c | leared. | |
| | С | If the addition gener | rates a carry, C is set; other | wise C is clea | ared. | | |
| | V | If an overflow occur | s, V is set; otherwise V is n | ot affected. | | | |
| | OVC | If (OVM = 0, disable then the counter is i overflow, then the c counter is not affect | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. | | | | |
| | OVM | If overflow mode b positive (0x7FFFFF overflowed. | it is set; then the ACC va FF) or maximum negative (0 | lue will satura 0x80000000) i | ate ma f the op | ximum eration | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the product shift value is a low bits are zero filled. I right shift operation), the up | he output oper s positive (log f the product oper bits are s | ration fr gical le shift v sign ext | om the ft shift alue is ended. | |
| Repeat | | If this operation is re The state of the Z, N an intermediate ov overflows if overflow | peated, then the instruction , C flags will reflect the final r erflow occurs. The OVC f v mode is disabled. | will be execut result. The V fla lag will count | ed N+1 ag will k interm | times. be set if rediate | |
| Example | ; Calc ; Y, M | ulate: Y = ((M*X > , X, B are Q15 val | > 4) + (B << 11)) >> 1 ues | 0 | | | |
| | SPM - | 4 | ; Set product shift | t to >> 4 | | | |
| | SETC S | XM | ; Enable sign exter | nsion mode | | | |
| | MOV T MPV P | , ലബ ന തx | ; T = M . P - M * Y | | | | |
| | MOV A | ,,,,,, CC,@B << 11 | ; ACC = S:B << 11 | | | | |
| | ADDL A | CC,P << PM | ; ACC = $(M*X >> 4)$ | + (S:B << 2 | 11) | | |
| | MOVH @ | Y,ACC << 5 | ; Store Q15 result | into Y | | | |

ADDL loc32,ACC

Add Accumulator to Specified Location

| Ś | SYNTAX C | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|----------------------------|---|---|---------------------------------|---------------------|--------------------|--|--|--|
| ADDL loc32, A | сс | | 0101 0110 0000 0001 0000 0000 LLLL LLLL | 1 | - | 1 | | | |
| Operands | loc32 | Addressing mode (s | ee Chapter 5) | | | | | | |
| | ACC | Accumulator registe | r | | | | | | |
| Description | | Add to the ACC regis "loc32" addressing m | Add to the ACC register the 32-bit content of the location pointed to by the "loc32" addressing mode: | | | | | | |
| | | [loc32] = [loc32] | + ACC; | | | | | | |
| | | This is a read-modify | his is a read-modify-write operation. | | | | | | |
| Flags and Modes | N | After the addition, th | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| | Z | After the addition, th | e Z flag is set if the ACC is | zero, else Z i | is clear | ed. | | | |
| | С | If the addition gener | f the addition generates a carry, C is set; otherwise C is cleared. | | | | | | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | | | | |
| | OVC | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. | | | | | | | |
| | OVM | If overflow mode bit (0x7FFFFFFF) or overflows. | is set, the ACC value will maximum negative (0x80 | saturate maxi 0000000) if t | mum p he ope | ositive eration | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only c | € RPT nce. | | | |
| Example | ; Incr MOVB A ADDL @ | ; Increment the 32-bit value VarA: MOVB ACC,#1 ; Load ACC with 0x0000001 ADDL @VarA,ACC ; VarA = VarA + ACC | | | | | | | |

ADDU ACC, loc16

Add Unsigned Value to Accumulator

| | SYNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|---|--|---|--|--------------------|--------------------|--|--|--|
| ADDU ACC, lo | c16 | | 0000 1101 LLLL LLLL | Х | Y | N+1 | | | |
| Operands | ACC | Accumulator registe | r | | | | | | |
| • | loc16 | Addressing mode (s | see Chapter 5) | | | | | | |
| | | . | . , | | | | | | |
| Description | | Add the 16-bit conte mode to the ACC reg add: | nts of the location pointed t ister. The addressed location | to by the "loc1 n is zero exten | 6" addr ded bef | essing ore the | | | |
| | | ACC = ACC + 0: [lc | oc16]; | | | | | | |
| Flags and Modes | Z | After the addition, the | After the addition, the Z flag is set if ACC is zero, else Z is cleared. | | | | | | |
| | Ν | After the addition, the | fter the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| | С | If the addition generates a carry, C is set; otherwise C is cleared. | | | | | | | |
| | v | If an overflow occurs | s, V is set; otherwise V is no | ot affected. | | | | | |
| | ovc | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation | | | | | | | |
| | ΟνΜ | If overflow mode bit is (0x7FFFFFFF) or overflowed. | s set; then the ACC value wi maximum negative (0x80 | ll saturate max 0000000) if t | timum p he op | ositive eration | | | |
| Repeat | | If this operation is repeated, then the instruction will be executed N+1 times. The state of the Z, N, C flags will reflect the final result. The V flag will be set if an intermediate overflow occurs. The OVC flag will count intermediate overflows, if overflow mode is disabled. | | | | | | | |
| Example | ; Add MOVU A ADD A ADDU A ADD A | three 32-bit unsig CC,@VarAlow CC,@VarAhigh << 16 CC,@VarBlow CC,@VarBhigh << 16 | ned variables by 16-bi ; AH = 0, AL = Va ; AH = VarAhigh, . ; ACC = ACC + 0:V ; ACC = ACC + Var | t parts: rAlow AL = VarAlow arBlow Bhigh << 16 Clow + Carry | J 7 | | | | |

ADD ACC,@VarChigh << 16 ; ACC = ACC + VarChigh << 16

ADDUL P,loc32

Add 32-bit Unsigned Value to P

| | SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|----------------|--|--|--|--------------------------------|------------------------------------|--|--|--|
| ADDUL P,loc | 32 | | 0101 0110 0101 0111 0000 0000 LLLL LLLL | 1 | _ | 1 | | | |
| Operands | Р | Product register | | | | | | | |
| | loc32 | Addressing mode (s | see Chapter 5) | | | | | | |
| Description | | Add to the P register addressing mode. T | the 32-bit content of the loca he addition is treated as an | tion pointed to unsigned AD | by the D opera | "loc32" ation: | | | |
| | | P = P + [loc32]; | = P + [loc32]; // unsigned add | | | | | | |
| | | Note: The difference be overflow count positive/negativ monitors the ca | between a signed and unsigned 3 ter (OVC). For a signed ADI e overflow. For an unsigned ADD, t rry. | 2-bit add is in the D, the OVC c he OVC unsigned | e treatme ounter d (OVCU | nt of the monitors) counter | | | |
| Flags and Modes | N | After the addition, if otherwise clear N. | bit 31 of the P register is 1, | then set the | N flag; | | | | |
| | Z | After the addition, i otherwise clear Z. | f the value of the P registe | er is 0, then s | et the | Z flag; | | | |
| | С | If the addition generation | ates a carry, set C; otherwis | e C is cleared | d. | | | | |
| | V | If an overflow occurs | s, V is set; otherwise V is no | ot affected. | | | | | |
| | OVCU | The overflow counte an unsigned carry. T | r is incremented when the a he OVM mode does not aff | ddition operat ect the OVCL | ion ger I count | ierates er. | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the | e RPT | | | |
| Example | ; Add 64 | -bit VarA + VarB a | and store result in Var | C: | | | | | |
| | MOVL P,@ | WarA+0 ; I | load P with low 32 bits | of VarA | | | | | |
| | MOVL ACC | C,@VarA+2 ; I | oad ACC with high 32 b | its of VarA | | | | | |
| | ADDUL P, | @VarB+0 ; A | dd to P unsigned low 3 | 2 bits of V | arB | | | | |
| | ADDCL AC | CC,@VarB+2 ; A | dd to ACC with carry h | igh 32 bits | of Va | rB | | | |
| | MOVL @Va | arC+0,P ; S | tore low 32-bit result | into VarC | | | | | |
| | MOVL @Va | arC+2,ACC ; S | store high 32-bit resul | t into VarC | | | | | |

ADDUL ACC, loc32

Add 32-bit Unsigned Value to Accumulator

| | SYNTAX | OPTIONS | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--------|--|--|---|---|---------------------------------|---------------------------------|--|--|
| ADDUL ACC, | loc32 | | 0101 xxxx | 0110 0101 0011 xxxx LLLL LLLL | 1 | Y | N+1 | | |
| Operands | ACC | Accumulator registe | er | | | - | | | |
| • | loc32 | Addressing mode (s | see Ch | anter 5) | | | | | |
| | 10002 | | | | | | | | |
| Description | | Add to the ACC regis by the "loc32" addres | ster the ssing n | unsigned 32-bit contender | ent of the loca | tion poi | nted to | | |
| | | ACC = ACC + [loc3] | 32]; | // unsigned add | | | | | |
| | | Note: The difference be overflow count positive/negative monitors the car | Note: The difference between a signed and unsigned 32-bit add is in the treatment of the overflow counter (OVC). For a signed ADD, the OVC counter monitors positive/negative overflow. For an unsigned ADD, the OVC unsigned (OVCU) counter monitors the carry. | | | | | | |
| Flags and Modes | z | After the addition, th | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | | | |
| | Ν | After the addition, th cleared. | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| | С | If the addition gener | rates a | carry, C is set; other | wise C is clea | red. | | | |
| | V | If an overflow occur | s, V is | set; otherwise V is n | ot affected. | | | | |
| | OVCU | The overflow counte an unsigned carry. 1 | er is inc The O∖ | remented when the a /M mode does not af | ddition operat | ion ger J coun | ierates ter. | | |
| Repeat | | If this operation is re The state of the Z, N an intermediate over | epeated I, C flag rflow o | l, then the instruction s will reflect the final r ccurs. The OVCU will | will be execut esult. The V fla count interme | ed N+1 ag will k ediate o | times. De set if Carries. | | |
| Example | ; Add | two 64-bit values | (VarA | and VarB) and st | ore result | in Va | rC: | | |
| · | MOVL | ACC,@VarA+0 | ; | Load ACC with con 32 bits of VarA | ntents of t | he low | ŗ | | |
| | ADDUL | ACC,@VarB+0 | ; | Add to ACC the co | ontents of | the lo | W | | |
| | MOVL | <pre>@VarC+0,ACC</pre> | ; | Store low 32-bit | result int | o VarC | 1 | | |
| | MOVL | ACC,@VarA+2 | ; | Load ACC with con 32 bits of VarA | ntents of t | he hig | jh | | |
| | ADDCL | ACC,@VarB+2 | ; | Add to ACC the co | ontents of | the hi | .gh | | |
| | MOVL | @VarC+2,ACC | ; | Store high 32-bi | t result in | to Var | C | | |

ADRK #8bit

Add to Current Auxiliary Register

| | SYNTAX C | OPTIONS | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|-------------------------|---|--|--|---------------------------------------|-----|-----|--|--|
| ADRK #8bit | | | 1111 | 1100 IIII IIII | Х | - | 1 | | |
| Operands | #8bit | 8-bit immediate cons | stant va | alue | | | | | |
| Description | | Add the 8-bit unsigned XAR (ARP) = XAR (AR | Add the 8-bit unsigned constant to the XARn register pointed to by ARP: AR(ARP) = XAR(ARP) + 0:8bit; | | | | | | |
| Flags and Modes | ARP | The 3-bit ARP points This pointer determin | he 3-bit ARP points to the current valid Auxiliary Register, XAR0 to XAR7. This pointer determines which Auxiliary register is modified by the operation. | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once | | | | | | | |
| Example | Table | A: .word 0x1111 word 0x2222 word 0x3333 word 0x4444 | | | | | | | |
| | FuncA : MOVI MOVZ | XAR1,#TableA AR2,*XAR1 | ;;;; | Initialize XAR1 Load AR2 with th pointed to by XA Set ARP = 1 | pointer e 16-bit va R1 (0x1111) | lue | | | |
| | ADRF MOVZ | C #2 Z AR3,*XAR1 | ; ; ; | Increment XAR1 b Load AR3 with th pointed to by XA | y 2 e 16-bit va R1 (0x3333) | lue | | | |

AND ACC,#16bit << #0..16

| S | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---|--|---|--|---------------------------------|------------------------------|--|--|
| AND ACC, #16 | bit << #0 | 15 | 0011 1110 0000 SHFT CCCC CCCC CCCC CCCC | 1 | - | 1 | | |
| AND ACC, #16 | bit << #16 | | 0101 0110 0000 1000 CCCC CCCC CCCC CCCC | 1 | - | 1 | | |
| Operands | ACC #16bit #016 | Accumulator registe 16-bit immediate cor Shift value (default is | Accumulator register 16-bit immediate constant value Shift value (default is "<< #0" if no value specified) | | | | | |
| Description | | Perform a bitwise AN unsigned constant va and lower order bits stored in the ACC reg ACC = ACC AND (0: | ND operation on the ACC realue left shifted as specified. are zero filled before the Algister: 16bit << shift value); | egister with the The value is z ND operation | e given zero exi . The re | 16-bit tended esult is | | |
| Flags and Modes | Ν | The load to ACC is tes negative flag bit is se | sted for a negative condition. et; otherwise it is cleared. | If bit 31 of ACC | C is 1, tł | nen the | | |
| | Z | The load to ACC is tested for a zero condition. The zero flag bit is set if the operation generates $ACC = 0$; otherwise it is cleared | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Calcu MOVL AG AND AG MOVL @W | alculate the 32-bit value: VarA = VarA AND 0x0FFFF000 L ACC,@VarA ; Load ACC with contents of VarA ACC,#0xFFFF << 12 ; AND ACC with 0x0FFFF000 L @VarA,ACC ; Store result in VarA | | | | | | |

AND ACC, loc16

| S | Ο ΧΑΤΝΥ | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---|--|---|--|----------------------|--------------------|--|--|
| AND ACC, loc1 | 6 | | 1000 1001 LLLL LLLL | 1 | Y | N+1 | | |
| Operands | ACC loc16 | Accumulator register Addressing mode (s | accumulator register addressing mode (see Chapter 5) | | | | | |
| Description | | Perform a bitwise AN content of the location stored in the ACC reg ACC = ACC AND 0:[| D operation on the ACC reg n pointed to by the "loc16" a gister: loc16] ; | ister with the z address mode | zero-ext . The re | tended esult is | | |
| Flags and Modes | N Z | Clear flag. The load to ACC is tested for a zero condition. The zero flag bit is set if the operation generates ACC = 0; otherwise it is cleared | | | | | | |
| Repeat | | This operation is repeatable. If the operation follows a RPT instruction, then the AND instruction will be executed $N+1$ times. The state of the Z and N flags will reflect the final result. | | | | | | |
| Example | ; Calcu MOVL AC AND AC MOVL @V | ulate the 32-bit va CC,@VarA CC,@VarB JarA,ACC | alue: VarA = VarA AND ; Load ACC with cont ; AND ACC with conte ; Store result in Va | D:VarB ents of Var nts of 0:Va rA | A rB | | | |

AND AX, loc16, #16bit

| | SYNTAX (| OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|--|--|---|---|-------------------|----------|--|
| AND AX, loc16 | 8, #16bit | | 1100 CCCC | 110A LLLL LLLL CCCC CCCC CCCC | Х | - | 1 | |
| Operands | AX | Accumulator high (A | H) or a | accumulator low (AL) | register | | | |
| | loc16 | Addressing mode (se | ee Cha | apter 5) | | | | |
| | #16bit | 16-bit immediate cor | stant | value | | | | |
| Description | | Perform a bitwise AND operation on the 16-bit contents of the location pointe to by the "loc16" addressing mode with the specified 16-bit immedia constant. The result is stored in the specified AX register: | | | | | | |
| | | AX = [loc16] AND 16bit; | | | | | | |
| Flags and Modes | N | The load to AX is tested for a negative condition. If bit 15 of AX is 1, then t negative flag bit is set; otherwise it is cleared. | | | | | | |
| | Z | The load to AX is test operation generates | sted fo AX = 0 | or a zero condition. T D; otherwise it is clear | he zero flag k red | oit is se | t if the | |
| Repeat | | This instruction is no instruction, it resets | ot repe the re | eatable. If this instruct peat counter (RPTC) | ion follows th and executes | e RPT s only c | nce. | |
| Example | ; Bran AND A SB D ; Merg ; VarC AND A AND A LSR A OR A MOV @ | ch if either of Bi L,@VarA,#0x0084 est,NEQ e Bits 0,1,2 of Va in bit locations L,@VarA,#0x0007 H,@VarB,#0x0700 H,#5 L,@AH VarC,AL | ts 2 ; rA wi 0,1,2 ; ; ; ; ; | and 7 of VarA are AL = VarA AND 0x0 Branch if result th Bits 8,9,10 of ,3,4,5: Keep bits 0,1,2 o Keep bits 8,9,10 Scale back bits 8 Merge bits Store result in V | non-zero: 0084 is non-zero VarB and st of VarA of VarB 3,9,10 to b: VarC | tore i its 3, | n 4,5 | |

AND IER,#16bit

Bitwise AND to Disable Specified CPU Interrupts

| | SYNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | | |
|--------------------|-----------------------------|--|---|--------------------------------|-------------------|-------|--|--|--|--|
| AND IER,#16k | bit | | 0111 0110 0010 0110 CCCC CCCC CCCC CCCC | Х | - | 2 | | | | |
| Operands | IER #16bit | Interrupt enable regi | Interrupt enable register | | | | | | | |
| | #1601 | 16-bit immediate cor | istant value (uxuuuu to uxf | FFF) | | | | | | |
| Description | | Disable specific interr register and the 16- register: | Disable specific interrupts by performing a bitwise AND operation with the IER egister and the 16-bit immediate value. The result is stored in the IER egister: | | | | | | | |
| | | IER = IER AND # | 16bit; | | | | | | | |
| Flags and Modes | | None | | | | | | | | |
| Repeat | | This instruction is no instruction, it resets | ot repeatable. If this instruct the repeat counter (RPTC) | ion follows th and executes | e RPT s only c | once. | | | | |
| Example | ; Disak ; inter AND 1 | ole INT1 and INT6 c rrupts enable: IER,#0xFFBE | only. Do not modify sta ; Disable INT1 and INT | te of other 36 | | | | | | |

AND IFR,#16bit

Bitwise AND to Clear Pending CPU Interrupts

| | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-----------------------------|--|---|---------------------------------|---------------------|----------|
| AND IFR,#16b | it | | 0111 0110 0010 1111 CCCC CCCC CCCC CCCC | х | - | 2 |
| Operands | IFR #16bit | Interrupt flag register 16-bit immediate constant value (0x0000 to 0xFFFF) | | | | |
| Description | | Clear specific pending interrupts by performing a bitwise AND operation with the IFR register and the 16-bit immediate value. The result of the AND operation is stored in the IFR register: | | | | |
| | | Note: Interrupt hardwa | are has priority over CPU instruction | on operation in | cases wl | here the |
| | | interrupt flag is b | being simultaneously modified by the | ne hardware and | the instr | uction. |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | € RPT |
| Example | ; Clear ; pendi AND I | the contents of the IFR register. Disables all ng interrupts: FR,#0x0000 ; Clear IFR register | | | | |

AND loc16, AX

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-------|---|--|---------------------------|-----------|-----------|--|
| AND loc16, AX | X | | 1100 000A LLLL LLLL | Х | - | 1 | |
| Operands | loc16 | Addressing mode (se | Addressing mode (see Chapter 5) | | | | |
| | AX | Accumulator high (Al | Accumulator high (AH) or accumulator low (AL) register | | | | |
| Description | | Perform a bitwise AND operation on the contents of the location pointed the "loc16" addressing mode with the specified AX register. The res stored in location pointed to by "loc16": | | | | | |
| | | [loc16] = [loc16] AND AX; | | | | | |
| | | This is a read-modif | y-write operation. | | | | |
| Flags and Modes | N | The load to [loc16] is then the negative flag | tested for a negative cond g bit is set; otherwise it is cl | ition. If bit 15 deared. | of [loc1 | 6] is 1, | |
| | Z | The load to [loc16] is operation generates | tested for a zero condition. ([loc16] = 0); otherwise it is | The zero flag cleared. | bit is se | et if the | |
| Repeat | | This instruction is no instruction, it resets | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | |
| Example | ; ANI |) the contents of Va | he contents of VarA with VarB and store in VarB: | | | | |
| | MOV | AL,@VarA | ; Load AL with cont | cents of Var | rA | | |
| | AND | @VarB,AL | ; VarB = VarB AND A | AL | | | |

AND AX, loc16

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--------------------------|--|---|--|---------------------|-----|--|
| AND AX, loc16 | 6 | | 1100 111A LLLL LLLL | Х | - | 1 | |
| Operands | АХ | Accumulator high (A | Accumulator high (AH) or accumulator low (AL) register | | | | |
| | loc16 | Addressing mode (se | Addressing mode (see Chapter 5) | | | | |
| Description | | Perform a bitwise AN with the 16-bit conter mode. The result is s AX = AX AND 16bit | Perform a bitwise AND operation on the contents of the specified AX register with the 16-bit contents of the location pointed to by the "loc16" addressing mode. The result is stored in the AX register: AX = AX AND 16bit; | | | | |
| Flags and Modes | N | The load to AX is tes negative flag bit is se | The load to AX is tested for a negative condition. If bit 15 of AX is 1, then the negative flag bit is set; otherwise it is cleared. | | | | |
| | Z | The load to AX is ter operation generates | The load to AX is tested for a zero condition. The zero flag bit is set if the operation generates $AX = 0$; otherwise it is cleared | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; AN MOV AND SB | D the contents of Va AL,@VarA AL,@VarB Dest,NEQ | arA and VarB and branch ; Load AL with con ; AND AL with cont ; Branch if result | if non-zer tents of Var ents of Var is non-zero | ro: rA B O | | |

AND loc16,#16bitSigned

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|--|---|---------------------------------|--------------------|------------------------------|--|
| AND loc16,#1 | 6bitSigned | | 0001 1000 LLLL LLLL CCCC CCCC CCCC CCCC | Х | - | 1 | |
| Operands | loc16 | Addressing m | Addressing mode (see Chapter 5) | | | | |
| | #16bitSigned | 16-bit signed | immediate constant value | | | | |
| Description | | Perform a bitwise AND operation on the 16-bit content of the pointed to by the "loc16" addressing mode and the speci immediate constant. The result is stored in the location point "loc16": | | | | ocation 16-bit d to by | |
| | | [loc16] = [loc16] AND 16bit; | | | | | |
| | | Smart Encoding: If loc16 = AH or AL and #16bitSigned is an 8-bit number, the assembler will encode this instruction as ANDB AX, #8-bit to im efficiency. To override this, use the ANDW AX, #16bitS instruction alias. | | | | | |
| Flags and Modes | N | After the operation | ation if bit 15 of [loc16] 1, s | et N; otherwis | e, clea | r N. | |
| | z | After the operation | ation if [loc16] is zero, set 2 | Z; otherwise, c | clear Z. | | |
| Repeat | | This instruction instruction, it once. | on is not repeatable. If this resets the repeat counter | instruction fol (RPTC) and (| lows th execute | e RPT s only | |
| Example | ; Clear Bits 3 and 11 of VarA: ; VarA = VarA AND #~(1 << 3 1 << 11) AND @VarA,#~(1 << 3 1 << 11) ; Clear bits 3 and 11 of VarA | | | | VarA | | |

ANDB AX, #8bit

Bitwise AND 8-bit Value

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|---|---|---|-----------|----------|--|
| ANDB AX, #8k | oit | | 1001 000A CCCC CCCC | Х | - | 1 | |
| Operands | AX #8bit | Accumulator high (Al 8-bit immediate cons | Accumulator high (AH) or accumulator low (AL) register 3-bit immediate constant value | | | | |
| Description | | Perform a bitwise AND operation with the content of the specified AX register (AH or AL) with the given 8-bit unsigned immediate constant zero extended. The result is stored in AX: AX = AX AND 0:8bit; | | | | | |
| Flags and Modes | N | The load to AX is tested for a negative condition. If bit 15 of AX is 1, then negative flag bit is set; otherwise it is cleared. | | | | ien the | |
| | Z | The load to AX is tested for a zero condition. The zero flag bit is set if the operation generates $AX = 0$; otherwise it is cleared | | | | t if the | |
| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Add MOV A ADD A ANDB MOV @ | VarA to VarB, keep L,@VarA L,@VarB AL,#0xFF VarC,AL | LSByte and store result ; Load AL with cont ; Add to AL content ; AND contents of A ; Store result in V | lt in VarC: cents of Var ts of VarB AL with 0x00 VarC | cA)FF | | |

Align Stack Pointer

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--------------|--|-------------------------------------|------------------------|---------|--------|
| ASP | | | 0111 0110 0001 1011 | Х | - | 1 |
| Operands | | None | | | | |
| Description | | Ensure that the stack pointer (SP) is aligned to an even address. If the lease significant bit of SP is 1, SP points to an odd address and must be moved be incrementing SP by 1. The SPA bit is set as a record of this alignment. Instead the ASP instruction finds that the SP already points to an even address, SP is left unchanged and the SPA bit is cleared to indicate that n alignment has taken place. In either case, the change to the SPA bit is made the decode 2 phase of the pipeline. | | | | |
| | | <pre>if(SP = odd) SP = SP + 1; SPA = 1;else SPA = 0;</pre> | | | | |
| | | If you wish to undo a j instruction. | previous alignment by the AS | SP instruction, | use the | NASP |
| Flags and Modes | SPA | If SP holds an odd ad cleared. | dress before the operation, S | SPA is set; oth | erwise, | SPA is |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | |
| Example | ; Alic | nment of stack p | oointer in interrupt | service ro | outine | 9: |
| | INTx: | .long INTxService | ; INTx interrup | t vector | | |
| | INTxSe: | rvice: | | | | |
| | ASP | | ; Align stack po | ointer | | |
| | NASP IRET | | ; Re-align stac ; Return from in | k pointer nterrupt. | | |

ASP

ASP

ASR AX,#1...16

Arithmetic Shift Right

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ASR AX,#116 | 1111 1111 101A SHFT | Х | - | 1 |

Operands AX Accumulator high (AH) or accumulator low (AL) register

1...16 Shift value

Description Perform an arithmetic right shift on the content of the specified AX register (AH or AL) by the amount given in the "shift value" field. During the shift, the value is sign extended and the last bit to be shifted out of the AX register is stored in the carry status flag bit:



| Flags and Modes | Ν | After the shift, if bit 15 c cleared. | of AX | is 1 then the negative flag bit is set; otherwise it is |
|--------------------|---|---|--------------------|--|
| | z | After the shift, if AX is | 0, th | en the Z bit is set; otherwise it is cleared. |
| | С | The last bit to be shifte | ed ou | it of AH or AL is stored in C. |
| Repeat | | This instruction is not instruction, it resets th | repe le re | eatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. |
| Example | ; Calc MOV <i>P</i> ADD <i>P</i> ASR <i>P</i> MOV @ | culate signed value: AL,@VarA AL,@VarB AL,#2 @VarC.AL | Var ; ; ; | C = (VarA + VarB) >> 2 Load AL with contents of VarA Add to AL contents of VarB Scale result by 2 Store result in VarC |

ASR AX,T

Arithmetic Shift Right

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ASR AX,T | 1111 1111 0110 010A | Х | - | 1 |

Operands AX Accumulator high (AH) or accumulator low (AL) register

T Upper 16 bits of the multiplicand (XT) register

Description Perform an arithmetic shift right on the content of the specified AX register as specified by the four least significant bits of the T register, T(3:0) = shift value = 0...15. The contents of higher order bits are ignored. During the shift, the value is sign extended. If the T(3:0) register bits specify a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of AX:



| Flags and Modes | Ν | After the shift, if bi cleared. Even if th AL is still tested fo | it 15 of A ne T(3:0) or the ne | X is 1 then the negative flag bit is set; otherwise it is) register bits specify a shift of 0, the value of AH or egative condition and N is affected. |
|--------------------|------|--|--------------------------------------|---|
| | Z | After the shift, if A T(3:0) register bits zero condition an | X is 0, th s specify d Z is af | en the Z bit is set, otherwise it is cleared. Even if the a shift of 0, the value of AH or AL is still tested for the fected. |
| | С | If T(3:0) specifies last bit to be shift | a shift o ed out o | f 0, then C is cleared; otherwise, C is filled with the f AH or AL. |
| Repeat | | This instruction is instruction, it res | s not rep ets the r | peatable. If this instruction follows the RPT repeat counter (RPTC) and executes only once. |
| Example | ; Ca | lculate signed val | lue: Va | rC = VarA >> VarB; |
| - | MOV | T,@VarB | ; | Load T with contents of VarB |
| | MOV | AL,@VarA | ; | Load AL with contents of VarA |
| | ASR | AL,T | ; | Scale AL by value in T bits 0 to 3 |
| | MOV | @VarC,AL | ; | Store result in VarC |

ASR64 ACC:P,#1..16

Arithmetic Shift Right of 64-bit Value

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| ASR64 ACC:P,#116 | 0101 0110 1000 SHFT | 1 | - | 1 |

Operands ACC:P Accumulator register (ACC) and product register (P)

- #1..16 Shift value
- **Description** Arithmetic shift right the 64-bit combined value of the ACC:P registers by the amount specified in the shift value field. As the value is shifted, the most significant bits are sign extended and the last bit shifted out is stored in the carry bit flag:



- Flags and
ModesNAfter the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and the
N bit is set; otherwise N is cleared.
 - **Z** After the shift, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared.
 - C The last bit shifted out of the combined 64-bit value is loaded into the C bit.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| ; Arithmetic shift right | the 64-bit Var64 by 10: |
|--------------------------|---|
| MOVL ACC,@Var64+2 | ; Load ACC with high 32 bits of Var64 |
| MOVL P,@Var64+0 | ; Load P with low 32 bits of Var64 |
| ASR64 ACC:P,#10 | ; Arithmetic shift right ACC:P by 10 |
| MOVL @Var64+2,ACC | ; Store high 32-bit result into Var64 |
| MOVL @Var64+0,P | ; Store low 32-bit result into Var64 |
| | ; Arithmetic shift right MOVL ACC,@Var64+2 MOVL P,@Var64+0 ASR64 ACC:P,#10 MOVL @Var64+2,ACC MOVL @Var64+0,P |

ASR64 ACC:P,T

Arithmetic Shift Right of 64-bit Value

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ASR64 ACC:P,T | 0101 0110 0010 1100 | 1 | - | 1 |

Operands ACC:P Accumulator register (ACC) and product register (P)

T Upper 16 bits of the multiplicand register (XT)

Description Arithmetic shift right the 64-bit combined value of the ACC:P registers by the amount specified in six least significant bits of the T register, T(5:0) = 0...63. Higher order bits are ignored. As the value is shifted, the most significant bits are sign extended. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC:P registers:



ACC:P

- Flags andNAfter the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and theModesN bit is set; otherwise N is cleared.
 - **Z** After the shift, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared.
 - **C** If (T[5:0] = 0) clear C; otherwise, the last bit shifted out of the combined 64-bit value is loaded into the C bit.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Arithmetic shift right | t the 64-bit Var64 by contents of Var16: |
|---------|--------------------------|--|
| | MOVL ACC,@Var64+2 | ; Load ACC with high 32 bits of Var64 |
| | MOVL P,@Var64+0 | ; Load P with low 32 bits of Var64 |
| | MOV T,@Var16 | ; Load T with shift value from Var16 |
| | ASR64 ACC:P,T | ; Arithmetic shift right ACC:P by T(5:0 |
| | MOVL @Var64+2,ACC | ; Store high 32-bit result into Var64 |
| | MOVL @Var64+0,P | ; Store low 32-bit result into Var64 |
| | | |

ASRL ACC,T

Arithmetic Shift Right of Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ASRL ACC,T | 0101 0110 0001 0000 | 1 | - | 1 |

| Operands | ACC | Accumulator register |
|----------|-----|----------------------|
| operanas | AUU | Accumulator register |

T Upper 16 bits of the multiplicand (XT) register

Description Perform an arithmetic shift right on the content of the ACC register as specified by the five least significant bits of the T register, T(4:0) = 0...31. Higher order bits are ignored. During the shift, the value is sign extended. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC register:



| Flags and | Z | After the shift, the Z flag is set if the ACC value is zero, else Z is cleared. Even |
|-----------|---|--|
| Modes | | if the T register specifies a shift of 0, the content of the ACC register is still |
| | | tested for the zero condition and Z is affected. |

- N After the shift, the N flag is set if bit 31 of the ACC is 1, else N is cleared. Even if the T register specifies a shift of 0, the content of the ACC register is still tested for the negative condition and N is affected.
- **C** If (T(4:0) = 0) then C is cleared; otherwise, the last bit shifted out is loaded into the C flag bit.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Arit | hmetic shift | right contents of VarA by VarB: |
|---------|--------|--------------|--|
| | MOVL | ACC,@VarA | ; ACC = VarA |
| | MOV | T,@VarB | ; T = VarB (shift value) |
| | ASRL | ACC,T | ; Arithmetic shift right ACC by T(4:0) |
| | MOVL | @VarA,ACC | ; Store result into VarA |
| | | | |

B 16bitOffset,COND

Branch

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|------------------|---|--|--|--------------|-------------------|-----|--|--|
| B 16bitOffset,COND | | | 1111 1111 1110 COND CCCC CCCC CCCC CCCC | Х | - | 7/4 | | | |
| Operands | 16bit- Offset | 16-bit s | igned imme | diate constant offset value (-32768 to +32767 range) | | | | | |
| | COND | Conditio | Conditional codes: | | | | | | |
| | | COND | Syntax | Description | Flags T | ested | | | |
| | | 0000 | NEQ | Not Equal To | Z = 0 | | | | |
| | | 0001 | EQ | Equal To | Z = 1 | | | | |
| | | 0010 | GT | Greater Then | Z = 0 | AND N | = 0 | | |
| | | 0011 | GEQ | Greater Then Or Equal T | ual To N = O | | | | |
| | | 0100 | LT | Less Then | N = 1 | | | | |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 | OR N = | = 1 | | |
| | | 0110 | HI | Higher | C = 1 | C = 1 AND $Z = 0$ | | | |
| | | 0111 | HIS, C | Higher Or Same, Carry S | Set $C = 1$ | | | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | | | |
| | | 1001 | LOS | Lower Or Same | C = 0 | OR Z = | = 1 | | |
| | | 1010 | NOV | No Overflow | V = 0 | | | | |
| | | 1011 | VO | Overflow | V = 1 | | | | |
| | | 1100 | NTC | Test Bit Not Set | TC = C |) | | | |
| | | 1101 | TC | Test Bit Set TC = 3 | | | | | |
| | | 1110 | NBIO | BIO Input Equal To Zero BIO = (| | 0 | | | |
| | | 1111 | UNC | Unconditional | - | | | | |
| Description | | Conditional branch. If the specified condition is true, then branch by adding the signed 16-bit constant value to the current PC value; otherwise continue execution without branching: | | | | | | | |
| | | <pre>If (COND = true) PC = PC + signed 16-bit offset; If (COND = false) PC = PC + 2; Note: If (COND = true) then the instruction takes 7 cycles. If (COND = false) then the instruction takes 4 avalage</pre> | | | | | | | |
| Flags and Modes | v | If the V flag is tested by the condition, then V is cleared. | | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |

BANZ 16bitOffset,ARn--

Branch if Auxiliary Register Not Equal to Zero

| SYNTAX OPTIONS | | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|---|-------------------------------|---|---------------------------------|---------------------|----------------|--|
| BANZ 16bitOff | fset,ARn−- | - | 0000 00 CCCC CC | 00 0000 1nnn CC CCCC CCCC | Х | - | 4/2 | |
| Operands | 16bit- Offset ARn | 16-bit signed immediate constant value Lower 16 bits of auxiliary registers XAR0 to XAR7 | | | | | | |
| Description | If the 16-bit content of the specified auxiliary register is not equal to 0, then the 16-bit sign offset is added to the PC value. This forces program control to the new address (PC + 16bitOffset). The 16-bit offset is sign extended to 22 bit before the addition. Then, the content of the auxiliary register is decremented by 1. The upper 16 bits of the auxiliary register (ARnH) is not used in the comparison and is not affected by the post decrement: | | | | | | | |
| | | if(ARn != 0) PC = PC + signe ARn = ARn - 1; ARnH = unchanged; | d 16-bit | offset; | | | | |
| | | Note: If branch is take If branch is not t | n, then the i aken, then t | nstruction takes 4 cycl he instruction takes 2 | es cycles | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repe the repea | atable. If this in t counter (RPTC) | struction follo and executes | ows the s only c | e RPT Ince. | |
| Example | ; Copy ; int3 ; int3 ; for(; Arr MOVL | <pre>the contents of A 2 Array1[N]; 2 Array2[N]; i=0; i < N; i++) ay2[i] = Array1[i] XAR2,#Array1 VAB2 #Array1</pre> | rrayl to ; ; | Array2: XAR2 = pointe | r to Arrayl | | | |
| | MOVL MOV Loop: | XAR3,#Array2 @AR0,#(N-1) | ; | Repeat loop N | r to Array2 times | | | |
| | MOVL MOVL BANZ | ACC, *XAR2++ *XAR3++, ACC Loop, AR0 | ; ; | ACC = Arrayl[Array2[i] = A Loop if AR0 ! | i] CC = 0, AR0 | | | |
BAR 16bitOffset,ARn,ARm,EQ/NEQ

Branch on Auxiliary Register Comparison

| | SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|--|---|---------------------|----------------|--|--|
| BAR 16bitOffs | et,ARn,ARm,EQ | 1000 1111 10nn nmmm CCCC CCCC CCCC CCCC | 1 | - | 4/2 | | |
| BAR 16bitOffs | et,ARn,ARm,NEQ | 1000 1111 11nn nmmm CCCC CCCC CCCC CCCC | 1 | - | 4/2 | | |
| | | | | | | | |
| Operands | 16bit- 16-bit signed imme Offset | ediate constant offset value | (-32768 to + | 32767 | range) | | |
| | ARn Lower 16 bits of au | uxiliary registers XAR0 to X | AR7 | | | | |
| | ARm Lower 16 bits of au | uxiliary registers XAR0 to X | AR7 | | | | |
| Syntax | Description | Condition | Tested | | | | |
| NEQ | Not Equal To | ARn != AF | lm m | | | | |
| EQ | Equal to | ARN = AR | m | | | | |
| Description | Compare the 16-bit contents of the two auxiliary registers ARn and ARm registers and branch if the specified condition is true; otherwise continue execution without branching: If (tested condition = true) PC = PC + signed 16-bit offset; If (tested condition = false) PC = PC + 2; Note: If (tested condition = true) then the instruction takes 4 cycles. If (tested condition = false) then the instruction takes 2 cycles. | | | | | | |
| Flags and Modes | None | | | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. | | |
| Example | ; String compare: MOVL XAR2,#StringA MOVL XAR3,#StringB MOV @AR4,#0 Loop: MOVZ AR0,*XAR2++ MOVZ AR1,*XAR3++ BAR Exit,AR0,AR4,EQ BAR Loop,AR0,AR1,EQ NotEqual: | ; XAR2 points t ; XAR3 points t ; AR4 = 0 ; AR0 = StringA ; AR1 = StringB ; Exit if Strin ; Loop if Strin ; StringA and B | o StringA o StringB [i] [i], i++ gA[i] = 0 gA[i] = Str not the sa | ingB[i me |] | | |
| | Exit: | ; StringA and B | the same | | | | |

BF 16bitOffset,COND

Branch Fast

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|---------------------|---------------------|---------|-----|-----|
| BF 16bitOffset,COND | 0101 0110 1100 COND | 1 | - | 4/4 |
| | CCCC CCCC CCCC CCCC | | | |

| Operands | 16bit- Offset | 16-bit si | gned imme | diate constant offset value (-327 | '68 to +32767 range) |
|--------------------|------------------|---|---|---|--|
| | COND | Conditio | nal codes: | | |
| | | COND | Syntax | Description | Flags Tested |
| | | 0000 | NEQ | Not Equal To | Z = 0 |
| | | 0001 | EQ | Equal To | Z = 1 |
| | | 0010 | GT | Greater Then | Z = 0 AND $N = 0$ |
| | | 0011 | GEQ | Greater Then Or Equal To | N = 0 |
| | | 0100 | LT | Less Then | N = 1 |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 OR N = 1 |
| | | 0110 | HI | Higher | C = 1 AND $Z = 0$ |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 |
| | | 1001 | LOS | Lower Or Same | C = 0 OR Z = 1 |
| | | 1010 | NOV | No Overflow | V = 0 |
| | | 1011 | OV | Overflow | V = 1 |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 |
| | | 1101 | TC | Test Bit Set | TC = 1 |
| | | 1110 | NBIO | BIO Input Equal To Zero | BIO = 0 |
| | | 1111 | UNC | Unconditional | - |
| Description | | Fast cor adding th continue If (COP If (COP If (COP If (COP If (COP) | nditional bra he signed 1 execution ND = true) ND = false branch fast e that reduce D = true) D = false | anch. If the specified condition is 6-bit constant value to the current without branching: PC = PC + signed 16-bit of e) PC = PC + 2; (BF) instruction takes advantage of d as the cycles for a taken branch from then the instruction takes) then the instruction takes | s true, then branch by nt PC value; otherwise ffset; ual prefetch queue on the 7 to 4: 4 cycles. 5 4 cycles. |
| Flags and Modes | v | If the V f | lag is tested | d by the condition, then V is clear | red. |
| Repeat | | This ins instructi | struction is on, it resets | not repeatable. If this instruc the repeat counter (RPTC) and | tion follows the RPT executes only once. |

C27MAP

Set the MOM1MAP Bit

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| C27MAP | 0101 0110 0011 1111 | Х | - | 5 |

Note: This instruction is an alias for the "CLRC M0M1MAP" operation.

Operands None

Clear the M0M1MAP status bit, configuring the mapping of the M0 and M1 Description memory blocks for C27x object-compatible operation. The memory blocks are mapped as follows:

| Program Space | | | Data Space | |
|---------------|----|--|------------|---------|
| | МО | | МО | 00 0000 |
| | M1 | | M1 | 00 0400 |





Note: The pipeline is flushed when this instruction is executed.

•

Flags and MOM1M The M0M1MAP bit is cleared. AP

Modes

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

; Set the device mode from reset to C27x object-compatible mode: Example Reset: C270BJ ; Enable C27x Object Mode ; Enable C27x/C28x Address Mode C28ADDR .c28 amode ; Tell assembler we are using C27x/C28x addressing C27MAP ; Enable C27x Mapping Of M0 and M1 blocks .

C27OBJ

Clear the OBJMODE Bit

| S | YNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|-------------------------------------|---------------------|--------------------|
| C27OBJ | | 0101 0110 0011 0110 | Х | - | 5 |
| Note: This instru | ction is an alias for the "CLRC C | DBJMODE" operation. | | | |
| Operands | None | | | | |
| Description | Clear the OBJM to execute C27x reset. | ODE status bit in Status Registe object code. This is the default | r ST1, configur mode of the p | ing the rocesso | device or after |
| | Note: The pipelin | ne is flushed when this instruction is exe | ecuted. | | |
| Flags and Modes | Clear the OBJM | ODE bit. | | | |
| Repeat | This instruction instruction, it re- | n is not repeatable. If this in sets the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT once. |
| Example | ; Set the device mode Reset: C270BJ ; E | e from reset to C27x: Snable C27x Object Mode | | | |
| | C28ADDR ; E .c28_amode ; T C27MAP ; E | Cnable C27x/C28x Address Mo Cell assembler we are in C Cnable C27x Mapping Of MO a | ode 27x/C28x add and M1 block | lr mode s | 9 |

C28ADDR

Clear the AMODE Status Bit

| SYNTAX OPTIONS | | | | OPCODE | OBJMODE | RPT | CYC |
|---------------------------|----------------------|-----------------------------------|------------------|---|---------------------------------|---------------------|----------------|
| C28ADDR | | 0101 0110 0001 0110 X - | | | | | |
| Note: This instruction is | an alias for t | he "CLRC AMOD | E" oper | ation. | | | |
| Operands | None | | | | | | |
| Description | Clear t C27x/0 | he AMODE s 28x addressi | tatus I ng mo | oit in Status Register de (see Chapter 5). | [·] ST1, putting | the de | vice in |
| | Note: | This instruction | does no | t flush the pipeline. | | | |
| Flags and AMOD Modes | E The AM | MODE bit is cl | eared. | | | | |
| Repeat | This i instruc | nstruction is ction, it resets | not re | epeatable. If this in peat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. |
| Example ; Exe ; C2x | cute the LP synta | operation x: | ″VarC | = VarA + VarB″ w | ritten in | | |
| LP | ADDR | | ; | Full C2xLP addres | ss compatibl | le mod | е |
| .1 | p_amode | | ; | Tell assembler w | e are in C22 | cLP mo | de |
| LD | P #VarA | | ; | Initialize DP (10 | ow 64K only) | | |
| LA | CL VarA | | ; | ACC = VarA (ACC) | nigh = 0) | | |
| AL | DS VarB | | ; | ACC = ACC + VarB | (unsigned) | | |
| SA CO | RADDR | | ; | Return to C28v a | ddregg mode | | |
| .0 | 28 amode | | ; | Tell assembler w | e are in C28 | 3x mod | e |

C28MAP

Set the MOM1MAP Bit

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| C28MAP | 0101 0110 0001 1010 | Х | - | 5 |

Note: This instruction is an alias for the "SETC M0M1MAP" instruction.

Data Space

MO

M1

00 0000

00 0400

00 07FF

Operands None

Description Set the M0M1MAP status bit in Status register ST1, configuring the mapping of the M0 and M1 memory blocks for C28x operation. The memory blocks are mapped as follows:



Program Space

MO

M1





Note: The pipeline is flushed when this instruction is executed.

Flags and MOM1MAP The MOM1MAP bit is set. Modes

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Set the device mode from reset to C28x mode: |
|---------|---|
| | Reset: |
| | C28OBJ ; Enable C28x Object Mode |
| | C28ADDR ; Enable C28x Address Mode |
| | .c28_amode ; Tell assembler we are in C28x address mode |
| | C28MAP ; Enable C28x Mapping Of M0 and M1 blocks |
| | |
| | |
| | |

C28OBJ

Set the OBJMODE Bit

| SYNTAX OPTIONS | | | 0 | PCODE | OBJMODE | RPT | CYC | |
|--|---------------|--|--|--|------------------------------------|---------------------------------|---------------------|----------------|
| C28OBJ 0101 0110 0001 1111 X | | | | | Х | - | 5 | |
| Note: This inst | ruction is an | alias for the "SET | C OBJMO | DDE" instruction | on. | - | | |
| Operands | | None | | | | | | |
| Description | | Set the OBJI (supports C2x | MODE kLP sou | status bit, irce): | putting the de | vice in C28x | object | mode |
| Flags and Modes | OBJ- MODE | Set the OBJM | 10DE b | it. | | | | |
| Repeat | | This instruct instruct instruction, it | tion is resets | not repeat | able. If this in counter (RPTC) | struction follo and executes | ows the s only c | e RPT once. |
| Example ; Set the device mode fr Reset: C280BJ ; Enal C28ADDR ; Enal .c28_amode ; Tel C28MAP ; Enal | | | om reset ole C28x (ole C27x/(l assemble ole C28x) | to C28x: Dbject Mode C28x Address er we are in Mapping Of M0 | Mode C27x/C28x a and M1 blo | ddress cks | s mode | |
| | | | | | | | | |

CLRC AMODE

Clear the AMODE Bit

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-------------------------------|--|--|--|-----------------|----------------|
| CLRC AMODE | | | 0101 0110 0001 0110 | Х | - | 1 |
| Operands | AMODE | Status bit | | | | |
| Description | | Clear the AMODE s addressing (see Ch | status bit in Status Register apter 5). | r ST1, enablir | ng C27) | x/C28x |
| | | Note: This instruction | on does not flush the pipelir | ne. | | |
| Flags and Modes | AMODE | The AMODE bit is o | cleared. | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in s the repeat counter (RPTC | struction follo | ows the | e RPT once. |
| Example | ; Execut ; syntax: SETC | te the operation | "VarC = VarA + VarB" w ; Full C2xLP addres | ritten in C ss-compatibl | 2xLP Le mode | e |
| | .lp_a | mode #VarA | ; Tell assembler we · Initialize DP () | e are in C23 | cLP mo | de |
| | LACL ADDS SACL CLRC | VarA VarB VarC AMODE amode | ; ACC = VarA (ACC F ; ACC = ACC + VarB ; Store result into ; Return to C28x ac ; Tell assembler we | <pre>inigh = 0) (unsigned) VarC ddress mode are in C28</pre> | 3x mod | e |
| | | | , | | | - |

CLRC MOM1MAP

Clear the MOM1MAP Bit

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| CLRC M0M1MAP | 0101 0110 0011 1111 | Х | - | 5 |

Operands MOM1MAP Status bit

Description Clear the M0M1MAP status bit in Status Register ST1, configuring the mapping of the M0 and M1 memory blocks for C27x operation. The memory blocks are mapped as follows:



Note: The pipeline is flushed when this instruction is executed. This bit is provided for compatibility for users migrating from C27x. The M0M1MAP bit should always remain set to 1 for users operating in C28x mode and C2xLP source-compatible mode.

Flags and MOM1MAP The MOM1MAP bit is cleared. Modes

Example ; Set the device mode from reset to C27x object-compatible mode: Reset: CLRC OBJMODE ; Enable C27x Object Mode CLRC AMODE ; Enable C27x/C28x Address Mode .c28_amode ; Tell assembler we are in C27x/C28x addr mode CLRC MOM1MAP ; Enable C27x Mapping Of M0 and M1 blocks . .

CLRC OBJMODE

Clear the OBJMODE Bit

| | SYNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|--|---|---|-------------------------|------------------|
| CLRC OBJMO | DE | | 0101 0110 0011 0110 | Х | - | 5 |
| Operands | OBJ- MODE | Status bit | | | | |
| Description | | Clear the OBJMODE status bit, enabling the device to execute C27x obj code. | | | | object |
| | | Note. The pipeline | | executed. | | |
| Flags and Modes | OBJ- MODE | The OBJMODE bit is cleared. | | | | |
| Repeat | | This instruction is instruction, it rese | s not repeatable. If this ir | nstruction follo C) and execut | ows the | e RPT / once. |
| Example | ; Set th Reset: CLRC C CLRC A .c28_a CLRC M | e device mode fr DEJMODE ; Ena MODE ; Ena mode ; Tel IOM1MAP ; Ena | om reset to C27x object ble C27x Object Mode ble C27x/C28x Address I l assembler we are in 0 ble C27x Mapping Of M0 | t-compatible Mode C27x/C28x a and M1 blo | e mode ddr mo cks | : de |

CLRC OVC

Clear Overflow Counter

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|--|---|---------------------------------|---------------------|----------------|--|
| CLRC OVC | | | 0101 0110 0101 1100 | 1 | - | 1 | |
| Note: This instr | uction is | an alias for the "ZAP OVC" op | eration. | | | | |
| Operands | ovc | Overflow counter bits in Status Register 0 (ST0) | | | | | |
| Description | | Clear the overflow counter (OVC) bits in ST0. | | | | | |
| Flags and Modes | ovc | The 6-bit overflow counter bits (OVC) are cleared. | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. | |
| Example | ; Cal CLRC MOVL ADDL ADDL SAT MOVL | .culate: VarD = sat(OVC ACC,@VarA ACC,@VarB ACC,@VarC ACC @VarD,ACC | <pre>VarA + VarB + VarC) ; Zero overflow cou ; ACC = VarA ; ACC = ACC + VarB ; ACC = ACC + VarC ; Saturate if OVC 1 ; Store saturated p</pre> | unter = 0 result into | VarD | | |

CLRC XF

Clear XF Status Bit

| ę | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|---|---|---------------------|---------------|
| CLRC XF | | | 0101 0110 0001 1011 | Х | - | 1 |
| Operands | XF | XF status bit and ou | tput signal | | | |
| Description | | Clear the XF status b | bit and pull the correspondir | ng output sign | al low. | |
| Flags and Modes | XF | The XF status bit is cleared. | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | ; Pulse MOV SB SETC CLRC Dest: | e XF signal high i AL,@VarA Dest,NEQ XF XF | f branch not taken: ; Load AL with cont ; ACC = VarA ; Set XF bit and si ; Clear XF bit and | ents of Var Ignal high signal low | сA | |
| | | | | | | |

CLRC Mode

Clear Status Bits

| | SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------|----------------|---------------------|---------|-----|------|
| CLRC | mode | 0010 1001 CCCC CCCC | Х | - | 1, 2 |
| CLRC | SXM | 0010 1001 0000 0001 | Х | - | 1 |
| CLRC | OVM | 0010 1001 0000 0010 | Х | - | 1 |
| CLRC | TC | 0010 1001 0000 0100 | Х | - | 1 |
| CLRC | С | 0010 1001 0000 1000 | Х | - | 1 |
| CLRC | INTM | 0010 1001 0001 0000 | Х | - | 2 |
| CLRC | DBGM | 0010 1001 0010 0000 | Х | - | 2 |
| CLRC | PAGE0 | 0010 1001 0100 0000 | Х | - | 1 |
| CLRC | VMAP | 0010 1001 1000 0000 | Х | - | 1 |

Description Clear the specified status bits. The "mode" operand is a mask value that relates to the status bits in this way:

| "Mode" bit | Status Register | Flag | Cycles |
|------------|-----------------|-------|--------|
| 0 | ST0 | SXM | 1 |
| 1 | ST0 | OVM | 1 |
| 2 | ST0 | тс | 1 |
| 3 | ST0 | С | 1 |
| 4 | ST1 | INTM | 2 |
| 5 | ST1 | DBGM | 2 |
| 6 | ST1 | PAGE0 | 1 |
| 7 | ST1 | VMAP | 1 |

Note: The assembler will accept any number of flag names in any order.

| Flags and | SXM | Any of the specified bits can be cleared by the instruction. |
|-----------|-------|--|
| Modes | OVM | |
| | тс | |
| | С | |
| | INTM | |
| | DBGM | |
| | PAGE0 | |
| | VMAP | |
| | | |

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Mod | ify flag settings: | | |
|---------|-------|--------------------|---|---------------------------------|
| | SETC | INTM,DBGM | ; | Set INTM and DBGM bits to 1 |
| | CLRC | TC,C,SXM,OVM | ; | Clear TC, C, SXM, OVM bits to 0 |
| | CLRC | #0xFF | ; | Clear all bits to O |
| | SETC | #0xFF | ; | Set all bits to 1 |
| | SETC | C,SXM,TC,OVM | ; | Set TC, C, SXM, OVM bits to 1 |
| | CLRC | DBGM, INTM | ; | Clear INTM and DBGM bits to 0 |

CMP AX, loc16

Compare

| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | |
|-----------------------|----------------------------|---|---|--|--------------------------|--|--|
| CMP AX, loc1 | 6 | | 0101 010A LLLL LLLL | Х | - | 1 | |
| Operands | AX loc16 | Accumulator high (Al Addressing mode (se | Accumulator high (AH) or accumulator low (AL) register Addressing mode (see Chapter 5) | | | | |
| Description | | The content of the specified AX register (AH or AL) is compared with the 16-bic content of the location pointed to by the "loc16" addressing mode. The result of (AX [loc16]) is evaluated and the status flag bits set accordingly. The AX register and content of the location pointed to by "loc16" are left unchanged. Set Flags On (AX - [loc16]); | | | | ♦ 16-bit ♦ result ♦ The AX anged: | |
| Flags and Modes | Ν | If the result of the operation is negative, then N is set; otherwise it The CMP instruction assumes infinite precision when it determines the result. For example, consider the subtraction 0x8000 – 0x0 precision were limited to 16 bits, the result would cause an over positive number 0x7FFF and N would be cleared. However, becaus instruction assumes infinite precision, it would set N to inc 0x8000 – 0x0001 actually results in a negative number | | | | leared. sign of . If the to the e CMP te that | |
| | z | The comparison is te operation (AX – [loc | ested for a zero condition. T 16]) = 0, otherwise it is clea | ⁻ he zero flag l ared. | oit is se | t if the | |
| | С | If the subtraction ger | nerates a borrow, then C is | cleared; other | wise C | is set. | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | ∍ RPT ince. | |
| Example | ; Bra MOV CMPB SB | nch if VarA is high AL,@VarA AL,@VarB Dest,HI | er then VarB: ; Load AL w ; Set Flags ; Branch if | ith content: On (AL - Va VarA highe: | s of V arB) r then | arA VarB | |

CMP loc16,#16bitSigned

Compare

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-----------------------|---|--|---|---|---|
| CMP loc16,#16 | bitSigned | | 0001 1011 LLLL LLLL CCCC CCCC CCCC CCCC | Х | - | 1 |
| Operands | loc16 #16bitSigned | Addressing m 16-bit immed | node (see Chapter 5) iate signed constant value | | | |
| Description | | Compare the addressing me perform the c evaluated and "loc16" is left | 16-bit contents of the location ode to the signed 16-bit im comparison, the result of (I the status flag bits are set unchanged: | on pointed to mediate cons [oc16] – #16 accordingly. T | by the ' tant va bitSign The cor | "loc16" lue. To ed)is itent of |
| | | Modify flags | s on ([loc16] - 16bitSi | lgned); | | |
| | | Smart Encodin If loc16 = AL assembler will this encoding, | ng: or AH and #16bitSigned is l encode this instruction as (use the CMPW AX, #16bit | an 8-bit num CMPB AX, #8I Signed instrue | nber, th pit, to o ction al | en the verride ias. |
| Flags and Modes | Ν | If the result of cleared. The C mines the sig 0x8000 – 0x0 would cause a be cleared. He precision, it we sults in a nega | the operation is negative, t CMP instruction assumes inf n of the result. For exampl 001. If the precision were I an overflow to the positive nu owever, because the CMP ould set N to indicate that 02 ative number. | hen N is set; inite precision e, consider th imited to 16 k umber 0x7FFF instruction as x8000 – 0x000 | otherwi when if he subt bits, the and N sumes 01 actu | ise it is t deter- raction e result I would infinite ally re- |
| | Z | The compariso | on is tested for a zero condit | ion. The zero | flag bit | is set if |
| | С | the operation If the subtracti set. | ([loc16] – 16bitSigned) = 0 on generates a borrow, then | , otherwise it C is cleared; | is clear otherwi | ed. se C is |
| Repeat | | This instructio instruction, it once. | on is not repeatable. If this resets the repeat counter | instruction fol (RPTC) and e | lows th execute | e RPT es only |
| Note: | | The example operating in (device into (OBJMODE b OBJMODE") | es in this chapter assume C28x Mode (OBJMODE = C28x mode following a res Dit in ST1 by executing instruction. | that the devi 1, AMODE = set, you mus the "C28OBJ | ce is a 0). To p t first s l" (or ' | already out the set the "SETC |

| Example | ; Calculate: | |
|---------|-------------------|-----------------------------|
| | ; if(VarA > 20) | |
| | ; VarA = 0; | |
| | CMP @VarA,#20 | ; Set flags on (VarA – 20) |
| | MOVB @VarA,#0,GT | ; Zero VarA if greater then |

| CMP64 ACC:P Compare 64-bit Value | | | | | Value | |
|----------------------------------|-------------|---|--|--|---|--|
| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
| CMP64 ACC: | Р | | 0101 0110 0101 1110 | 1 | - | 1 |
| Operands | ACC:P | Accumulator register | r (ACC) and product registe | er (P) | | |
| Description | | The 64-bit content of and the flags are set | the combined ACC:P registe appropriately: | ers is compare | d agair | ist zero |
| | | <pre>if((V = 1) & (ACC N = 0; else N = 1; if((V = 1) & (ACC N = 1; else N = 0; if(ACC:P = 0x8000 Z = 1; else Z = 0; V = 0; Note: This operation sl CMP64 ACC:P ; Cle perform 64-bit oper CMP64 ACC:P ; Set conditionally brance</pre> | <pre>C(bit 31) = 1)) C(bit 31) = 0)) 0 0000 0000 0000) hould be used as follows: ar V flag ation Z,N flags, V=0 h</pre> | | | |
| Flags and Modes | N Z V | The content of the A value is negative. Th overflow flag (V) to in For example, consid 0001. This results in would be set. Beca overflow, it would into number. If the value i is cleared. The zero flag bit is se cleared. The state of the V flag if the value in the AC | ACC register is tested to de the CMP64 instruction takes in crease precision when deted der the subtraction on ACC an overflow to a positive nu ause the CMP64 instruction erpret the result as a negative is ACC is found to be negative et if the combined 64 bits of A the subtraction of the subtraction of the CP register is posetive V is | termine if the into account the ermining if AC of 0x8000 0 mber (0x7FFF on takes into re number and re, then N is se ACC:P is zero, the ACC registe | 64-bit he state C is ne 000 – (FFFF) accou I not a p t; other otherw | ACC:P \Rightarrow of the \Rightarrow gative. \Rightarrow 0000 \Rightarrow and V \Rightarrow 0000 \Rightarrow 00000 \Rightarrow 00000 \Rightarrow 00000 \Rightarrow 00000 \Rightarrow 00000 \Rightarrow 00000 \Rightarrow 000000 \Rightarrow 000000 \Rightarrow 000000 \Rightarrow 000000000000000000000000000000000000 |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in: the repeat counter (RPTC) | s cleared by t struction follo and executes | ows the only o | ation. e RPT nce. |

| Example | ; If 64-bit VarA > | 64-bit VarB, branch: |
|---------|--------------------|---|
| | CMP64 ACC:P | ; Clear V flag |
| | MOVL P,@VarA+0 | ; Load P with low 32 bits of VarA |
| | MOVL ACC,@VarA+2 | ; Load ACC with high 32 bits of VarA |
| | SUBUL P,@VarB+0 | ; Sub from P unsigned low 32 bits of VarB |
| | SUBBL ACC,@VarB+2 | ; Sub from ACC with borrow high 32 bits of VarB |
| | CMP64 ACC:P | ; Set Z,N flags appropriately for ACC:P |
| | SB Dest,GT | ; branch if VarA > VarB |
| | | |

CMPB AX, #8bit

Compare 8-bit Value

| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|-----------------------|--|---|---|--|---------------------|-----|--|--|--|
| CMPB AX, #8 | bit | | 0101 001A CCCC CCCC | Х | - | 1 | | | |
| Operands | AX #8bit | Accumulator high (AH) or accumulator low (AL) register 8-bit immediate constant value | | | | | | | |
| Description | | Compare the content of the specified AX register (AH or AL) with the zero-extended 8-bit unsigned immediate constant. The result of (AX – 0:8bit) is evaluated and the status flag bits are set accordingly. The content of the AX register is left unchanged: Set Flags On (AX – 0:8bit); | | | | | | | |
| Flags and Modes | Ν | If the result of the operation is negative, then N is set; otherwise it is cleared. The CMPB instruction assumes infinite precision when it determines the sign of the result. For example, consider the subtraction 0x8000 – 0x0001. If the precision were limited to 16 bits, the result would cause an overflow to the positive number 0x7FFF and N would be cleared. However, because the CMPB instruction assumes infinite precision, it would set N to indicate that 0x8000 – 0x0001 actually results in a negative number. | | | | | | | |
| | z | The comparison is tested for a zero condition. The zero flag bit is set if the operation (AX – [0:8bit]) = 0, otherwise it is cleared. | | | | | | | |
| | С | If the subtraction generates a borrow, then C is cleared; otherwise C is set. | | | | | | | |
| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | | |
| Example | ; Chec MOV A CMPB SB C CMPB SB C | ck if VarA is withi AL,@VarA AL,#0xF0 DutOfRange,GT AL,#0x80 DutOfRange,LT | n range 0x80 <= VarA < ; Load AL with cont ; Set Flags On (AL ; Branch if VarA gr ; Set Flags On (AL ; Branch if VarA le | = 0xF0: tents of Van - 0x00F0) reater then - 0x0080) ess then 0x0 | cA 0x00F 0080 | F | | | |

CMPL ACC, loc32

Compare 32-bit Value

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|-----------------------|--|---|---|-----------------------------------|--------|-----|--|--|
| CMPL ACC,loc32 | | | 0000 1111 LLLL LLLL | Х | - | 1 | | |
| Operands | ACC loc32 | Accumulator register Addressing mode (see Chapter 5) | | | | | | |
| Description | | The content of the ACC register is compared with the 32-bit location pointed to by the "loc32" addressing mode. The status flag bits are set according to the result of (ACC – [loc32]). The ACC register and the contents of the location pointed to by "loc32" are left unchanged: Modify flags on (ACC – [loc32]); | | | | | | |
| Flags and Modes | Ν | If the result of the operation is negative, then N is set; otherwise it is cleared. The CMPL instruction assumes infinite precision when it determines the sign of the result. For example, consider the subtraction 0x8000 0000 – 0x0000 0001. If the precision were limited to 32 bits, the result would cause an overflow to the positive number 0x7FFF FFFF and N would be cleared. However, because the CMPL instruction assumes infinite precision, it would set N to indicate that 0x8000 0000 – 0x0000 0001 actually results in a negative number. | | | | | | |
| | Z | The comparison is tested for a zero condition. The zero flag bit is set if the operation $(AX - [loc32]) = 0$, otherwise it is cleared. | | | | | | |
| | С | If the subtraction ge | nerates a borrow, C is clear | red; otherwise | C is s | et. | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Swap MOVL A MOVL P CMPL A MOVL @ MOVL @ | the contents of 3: CC,@VarB ,@VarA CC,@P VarA,ACC,HI P,ACC,HI VarA,P | 2-bit VarA and VarB if ; ACC = VarB ; P = VarA ; Set flags on (Var ; VarA = ACC if higher ; VarA = P | VarB is hig rB - VarA) gher | gher: | | | |

CMPL ACC,P << PM

Compare 32-bit Value

| SYNTAX OPTIONS | | | OPCODE | | | OBJMODE | RPT | CYC | |
|-----------------------|---|--|--|--|--|---|--|--|--|
| CMPL ACC,P << PM | | | 1111 | 1111 | 0101 | 1001 | Х | - | 1 |
| Operands | ACC | Accumulator registe | r | | | | | | |
| | Р | Product register | | | | | | | |
| | < <pm< th=""><th>Product shift mode</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></pm<> | Product shift mode | | | | | | | |
| Description | | The content of the ACC register is compared with the content of the P reshifted by the amount specified by the product shift mode (PM). The flag bits are set according to the result of (ACC –[P << PM]). The conten ACC register and the P register are left unchanged: | | | | | | egister, status nt of the | |
| | | Modify flags on (| ACC - | [P < | < PM] |); | | | |
| Flags and Modes | Ν | If the result of the op The CMPL instructio of the result. For exa 0001. If the precisio overflow to the pos However, because th set N to indicate th negative number. | eration on assu ample, on wer itive nu he CM at 0x8 | n is ne imes i consi e limi umbei PL ins 000 (| gative, nfinite der the ted to r 0x7F tructio 0000 - | then N is precision subtrac 32 bits, FF FFF n assume 0x0000 | s set; otherwis when it deten tion 0x8000 0 the result wo and N woul es infinite pred 0001 actuall | e it is c mines t 000 – (ould ca d be c cision, i y resul | leared. he sign 0x0000 use an leared. t would lts in a |
| | Z | The comparison is t operation (AX – [P< | The comparison is tested for a zero condition. The zero f operation $(AX - [P << PM]) = 0$, otherwise, it is cleared. | | | | | bit is se | et if the |
| | С | If the subtraction ge | nerate | s a bo | orrow, | C is clea | eared; otherwise C is set. | | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits se the pi e low b right s | ts the roduc oits ar hift op | shift m t shift e zero peratio | node for th value is filled. If n), the up | he output oper positive (log f the product oper bits are s | ration fr gical le shift v sign ext | rom the off shift alue is rended. |
| Repeat | | This instruction is not repeatable. If this instruction follows the instruction, it resets the repeat counter (RPTC) and executes only on | | | | | e RPT once. | | |
| Example | ; Comp MOVL SPM MOVL | are the following ACC,@VarA -4 P,@VarB | (VarA | - Va ; AC ; Se ; P | rB >> C = V t the = Var | 4): arA product B | t shift mod | e to " | >> 4″ |
| | CMPL | ACC,P << PM | | ; Co | mpare | (VarA · | – VarB >> 4 |) | |

CMPR 0/1/2/3

Compare Auxiliary Registers

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| CMPR 0 | 0101 0110 0001 1101 | 1 | - | 1 |
| CMPR 1 | 0101 0110 0001 1001 | 1 | - | 1 |
| CMPR 2 | 0101 0110 0001 1000 | 1 | - | 1 |
| CMPR 3 | 0101 0110 0001 1100 | 1 | - | 1 |

Operands

None

DescriptionCompare AR0 to the 16-bit auxiliary register pointed to by ARP. The
comparison type is determined by the instruction.CMPR 0: if (AR0 = AR [ARP]) TC = 1, else TC = 0

- Flags andARPThe 3-bit ARP points to the current valid Auxiliary Register, XAR0 to XAR7.ModesThis pointer determines which Auxiliary register is compared to AR0.
 - TC If the test is true, TC is set, else TC is cleared.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | TableA: .word 0x1111 | |
|---------|----------------------|-------------------------------------|
| | .word 0x2222 | |
| | FuncA: | |
| | MOVL XAR1,#VarA | ; Initialize XAR1 Pointer |
| | MOVZ AR0,*XAR1++ | ; Load AR0 with 0x1111, clear AR0H, |
| | | ; ARP = 1 |
| | MOVZ AR1,*XAR1 | ; Load AR1 with 0x2222, clear AR1H |
| | CMPR 0 | ; AR0 = AR1? No, clear TC |
| | B Equal,TC | ; Don't branch |
| | CMPR 2 | ; AR1 > AR2? Yes, set TC |
| | B Less,TC | ; Branch to "Less" |
| | | |

CSB ACC

Count Sign Bits

| SYNIAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|----------------|---------|-----|-----|
| CSB ACC 0101 | 0110 0011 0101 | 1 | - | 1 |

| Operands | ACC | Accur | nulator register | |
|--------------------|--------|-------------------|---|--|
| Description | | Count or 1s ii | the sign bits in the AC n the ACC register ar | C register by determining the number of leading 0s nd storing the result, minus one, in the T register |
| | | T = 0 T = 1 | , 1 sign bit , 2 sign bits | |
| | | T = 31 | l, 32 sign bits | |
| | | Note: | The count sign bit operati useful for algorithms suc inverse of a number, sea | ion is often used in normalization operations and is particularly ch as; calculating Square Root of a number, calculating the arching for the first "1" bit in a word. |
| Flags and Modes | N | N is s | et if bit 31 of ACC is [.] | 1, else N is cleared. |
| | z | Z is se | et if ACC is 0, else Z | is cleared. |
| | тс | The T negat | C bit will reflect the s ve). | state of the sign bit after the operation (TC=1 for |
| Repeat | | This instrue | instruction is not re ction, it resets the rep | epeatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. |
| Example | ; Norm | nalize | the contents of V | /arA: |
| - | MOVL | ACC,@ | VarA | ; Load ACC with contents of VarA |
| | CSB | ACC | | ; Count sign bits |
| | MOVL | @VarA | , ACC | ; Store result into VarA |
| | | | | |

DEC loc16

Decrement by 1

| | SYNTA | K OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | | |
|-------------|-------|---|--|---------|-----|-----|--|--|--|--|
| DEC loc16 | | | 0000 1011 LLLL LLLL | Х | - | 1 | | | | |
| Operands | loc16 | Addressing mode (s | Addressing mode (see Chapter 5) | | | | | | | |
| Description | | Subtract 1 from the signed content of the location pointed to by the "loc1 addressing mode: | | | | | | | | |
| Flags and | N | After the operation if bit 15 of [loc16] is 1, set N; otherwise, cl | | | | | | | | |
| Modes | z | After the operation it | After the operation if [loc16] is zero, set Z; otherwise, clear Z. | | | | | | | |
| | С | If the subtraction ge | If the subtraction generates a borrow, C is cleared; otherwise C is set. | | | | | | | |
| | v | If an overflow occurs, V is set; otherwise V is not affected. | | | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the instruction, it resets the repeat counter (RPTC) and executes only or | | | | | | | | |
| Example | | ; Va | arA = VarA - 1; | | | | | | | |
| | DEC | @VarA ; De | crement contents of | VarA | | | | | | |

Disable Maskable Interrupts (Set INTM Bit)

| S | YNTAX OPTIC | NS | | OP | CODE | | OBJMODE | RPT | CYC |
|--------------------|--|--|-----------------------------|--|--|---|---|---------------------|---------------|
| DINT | | | 0011 | . 1011 | 0001 00 | 000 | Х | - | 2 |
| Note: This instru | uction is an alias | for the "SETC r | node" operat | tion with | the "mode" | field = I | NTM. | | |
| Operands | No | ne | | | | | | | |
| Description | Dis no | able all masl effect on the | kable CPU unmaska | interru ble res | pts by set et or NM | ting the | e INTM status upts. | bit. DII | NT has |
| Flags and Modes | INTM The | The instruction sets this bit to disable interrupts. | | | | | | | |
| Repeat | Thi ins | s instructior truction, it re | n is not r sets the re | epeata epeat c | able. If th counter (F | nis ins IPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | ; Make the DINT MOVL ADDL MOVL EINT | operation ACC,@VarA ACC,@VarB @VarC,ACC | "VarC = ; ; ; ; | VarA Disa ACC ACC Stor Enab | + VarB" ble inte = VarA = ACC + e result le inter | atomi errupt VarB : into rrupts | c: s (INTM = 1 VarC (INTM = 0) | _) | |

DMAC ACC:P,loc32,*XAR7/++

16-Bit Dual Multiply and Accumulate

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------------|---------------------|---------|-----|-----|
| DMAC ACC:P,loc32,*XAR7 | 0101 0110 0100 1011 | 1 | Y | N+2 |
| | 1100 0111 LLLL LLLL | | | |
| DMAC ACC:P,loc32,*XAR7++ | 0101 0110 0100 1011 | 1 | Y | N+2 |
| | 1000 0111 LLLL LLLL | | | |

Operands ACC:P Accumulator register (ACC) and product register (P)

loc32 Addressing mode (see Chapter 5)

- **Note:** The @ACC and @P register addressing modes cannot be used. No illegal instruction trap will be generated if used (assembler will flag an error).
- *XAR7 /++ Indirect program-memory addressing using auxiliary register XAR7, can access full 4M x 16 program space range (0x000000 to 0x3FFFFF)

Description

Dual 16-bit x 16-bit signed multiply and accumulate. The first multiplication takes place between the upper words of the 32-bit locations pointed to by the "loc32" and *XAR7/++ addressing modes and second multiplication takes place with the lower words.



After the operation the ACC contains the result of multiplying and adding the upper word of the addressed 32-bit operands. The P register contains the result of multiplying and adding the lower word of the addressed 32-bit operands.

| XT | = [loc32]; | |
|------|---|--|
| Temp | = Prog[*XAR7 or *XAR7++]; | |
| ACC | <pre>= ACC + (XT.MSW * Temp.MSW) << PM;</pre> | |
| Р | = P + (XT.LSW * Temp.LSW) << PM; | |

Z, N, V, C flags and OVC counter are affected by the operation on ACC only. The PM shift affects both the ACC and P operations.

On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*XAR7/++" addressing mode can be used to access data space variables that fall within the program space address range.

| | | With some addressing mode combinations, you can get conflicting references. In such cases, the C28x will give the "loc16/loc32" field priority on changes to XAR7. For example: |
|--------------------|-----|--|
| | | DMAC ACC:P,*XAR7,*XAR7++ ;XAR7 given priority DMAC ACC:P,*XAR7++,*XAR7 ; *XAR7++ given priority DMAC ACC:P,*XAR7,*XAR7++ ; *XAR7++ given priority |
| Flags and Modes | z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. |
| | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | С | If the addition generates a carry of the ACC register, C is set; otherwise C is cleared. |
| | V | If an overflow of the ACC register occurs, V is set; otherwise V is not affected. |
| | OVC | If overflow mode is disabled; and if the operation generates a positive overflow of the ACC register, then the counter is incremented. If overflow mode is disabled; and if the operation generates a negative overflow of the ACC register, then the counter is decremented. |
| | ΟνΜ | If overflow mode bit is set; then the ACC value will saturate maximum pos- itive (0x7FFFFFF) or maximum negative (0x80000000) if the operation overflowed. Note that OVM only affects the ACC operation. |
| | РМ | The value in the PM bits sets the shift mode for the output operation from the product register. The PM mode affects both the ACC and P register accumulates. If the product shift value is positive (logical left shift opera- tion), then the low bits are zero filled. If the product shift value is negative (arithmetic right shift operation), the upper bits are sign extended. |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed N+1 times. The state of the Z, N, C and OVC flags will reflect the final result in the ACC. The V flag will be set if an intermediate overflow occurs in the ACC. |

| Example | <pre>; Calcu ; int10 ; int10 ; ; ; ; ; ; sum = ; ; for (2)</pre> | <pre>alate sum of 5 X[N] 5 C[N] = 0; 6 0; 5 0;</pre> | product ; ; Data ir ; Coeffic ; Data ar ; N must | ormat ormat ent i Coef e an | dual 16-bit ion nformation f must be a even number | multiply: (located in l ligned to eve | .ow 4M) en address |
|---------|--|--|--|---|--|---|-----------------------|
| | ; 101(1 ; sum = | = 0; 1 < N; = sum + (X[i | 1++)] * C[i]) | >> 5: | | | |
| | MOVL | XAR2,#X | | XAR2 | = pointer | to X | |
| | MOVL | XAR7,#C | | XAR7 | = pointer | to C | |
| | SPM | -5 | | Set | - product shi | ft to ">> 5" | |
| | ZAPA | | | Zero | ACC, P, OV | Ċ | |
| | RPT | #(N/2)-1 | | Repe | at next ins | truction $N/2$ | times |
| | DMAC | P,*XAR2++,* | XAR7++ | ACC P = | = ACC + (X[P + (X[i] * | i+1] * C[i+1] C[i]) >> 5 i |) >> 5 .++ |
| | ADDL | ACC,@P | | Perf | orm final a | ccumulate | |
| | MOVL | @sum,ACC | | Stor | e final res | ult into sum | |
| | | | | | | | |

DMOV loc16

Data Move Contents of 16-bit Location

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|---|---|---------------------|---------------------|
| DMOV loc16 | | | 1010 0101 LLLL LLLL | 1 | Y | N+1 |
| Operands | loc16 | Addressing mode Note: For this ope @ARn, @A generated. | e (see Chapter 5) ration, register-addressing modes o H, @AL, @PH, @PL, @SP, @T. A | annot be used. An illegal instruc | The mo tion trap | des are: will be |
| Description | | Copy the content | s pointed to by "loc16" into the | e next highest | addres | ss: |
| | | [loc16 + 1] = | [loc16]; | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is then it will be exe | repeatable. If the operation is cuted N+1 times. | s follows a RF | PT instr | uction, |
| Example | ; Calcu ; intle ; intle ; Y = (; X[2] ; X[1] SPM MOVP MOVA MPY MOVA MPY ADDL DMOV DMOV | <pre>alate using 16-b 5 X[3]; 5 C[3]; (X[0]*C[0] >> 2) = X[1]; = X[0]; -2 ; T,@X+2 ; P,T,@C+2 ; T,@X+1 ; P,T,@C+1 ; T,@X+0 ; P,T,@C+0 ; ACC,P << PM ; @X+1 ; @X+0 ; @X ACC</pre> | <pre>it multiply: + (X[1]*C[1] >> 2) + (X Set product shift to >> T = X[2] P = T*C[2], ACC = 0 T = X[1], ACC = X[2]*C[2 P = T*C[1] T = X[0], ACC = ACC + X P = T*C[0] ACC = ACC + X[0]*C[0] >> X[2] = X[1] X[1] = X[0] Store regult into X</pre> | <pre>[2] *C[2] >> 2 2 2] >> 2 [1] *C[1] >> > 2</pre> | 2); | |

EALLOW

Enable Write Access to Protected Space

| 5 | SYNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|--|--|--|--|--|
| EALLOW | | | 0111 0110 0010 0010 | Х | - | 4 |
| Operands | | None | | | | |
| Description | | Enable access to e | emulation space and other p | protected regi | sters. | |
| | | This instruction sets set, the C28x CPU well as other prote device to determine | s the EALLOW bit in status r allows write access to the m acted registers. See the da e which registers the EALL | egister ST1. V emory-mappe ta sheet for y OW bit protec | Vhen th ed regis rour pa ts. | is bit is ters as rticular |
| | | To again protect aç | gainst writes to the registers | s, use the ED | IS instr | uction. |
| | EALLOW only controls write access; reads are allowed even if E has not been executed. | | | | en if EA | LLOW |
| | | On an interrupt or tr the stack within S Therefore, at the st registers is disable the EALLOW bit sa | rap, the current state of the E ST1 and the EALLOW bit art of an interrupt service rou d. The IRET instruction will aved on the stack. | ALLOW bit is t is autocrati utine access to restore the c | saved of cally c the pro urrent s | off onto leared. otected state of |
| | | The EALLOW bit is overridden via the JTAG port, allowing full control of register accesses during debug from Code Composer Studio. | | | | |
| Flags and Modes | EALLOW | The EALLOW flag | is set. | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in sthe repeat counter (RPTC | struction follo | ows the | e RPT once. |
| Example | ; Enable EALL AND MOV EDIS | access to RegA OW @RegA,#0x4000 @RegB,#0 | and RegB which are EAL ; Enable access to sel ; RegA = RegA AND 0x04 ; RegB = 0 ; Disable access to se | LOW protect ected regis 00 elected regi | ed: sters isters | |

EDIS

Disable Write Access to Protected Registers

| S | SYNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|------------------------|---|--|---|--------------------------------|----------------|--|
| EDIS | | | 0111 0110 0001 1010 | Х | - | 4 | |
| Operands | | None | | | | | |
| Description | | Disable access to | emulation space and other | protected reg | isters. | | |
| | | This instruction clears the EALLOW bit in status register ST1. When this bit is clear, the C28x CPU does not allow write access to the memory–mapped emulation registers and other protected registers. See the data sheet for your particular device to determine which registers the EALLOW bit protects. | | | | | |
| | | To allow write acce | ess to the registers, use the | EALLOW ins | tructior | ۱. | |
| Flags and Modes | EALLOW | The EALLOW flag | is cleared. | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in sthe repeat counter (RPTC | struction follo | ows the | e RPT once. | |
| Example | ; Enabl EALL NOP | e access to Reg OW | A and RegB which are ; Enable access to sel ; Wait 2 cycles for en ; effect. The number c ; and/or register depe | e EALLOW pr ected regis mable to tak of cycles is endant. | rotect sters se devid | ced: | |
| | AND MOV EDIS | @RegA,#0x4000 @RegB,#0 | ; RegA = RegA AND 0x04 ; RegB = 0 ; Disable access to se | 00 elected regi | .sters | | |

Enable Maskable Interrupts (Clear INTM Bit)

| | | | | | | 1 | T | |
|--------------------|--|--|----------------------------|--|---|------------------------------------|---------------------|----------------|
| w | Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο | DNS | | OPC | ODE | OBJMODE | RPT | CYC |
| EINT | | | 0010 | 1001 | 0001 0000 | X | - | 2 |
| | Note | : This instruc | tion is an alias | s for the ' | CLRC mode" op | eration with the "m | node" fiel | d = INTM. |
| Operands | No | ne | | | | | | |
| Description | En | able interrup | ts by cleari | ing the | INTM status I | oit. | | |
| Flags and Modes | INTM Th | This bit is cleared by the instruction to enable interrupts. | | | | | | |
| Repeat | Th | is instructior truction, it re | n is not re sets the re | epeatab peat co | ole. If this ir unter (RPTC | nstruction follo) and executes | ows the s only c | e RPT ince. |
| Example | ; Make the DINT MOVL ADDL MOVL FINT | operation ACC,@VarA ACC,@VarB @VarC,ACC | "VarC = ; ; ; ; | VarA + Disab ACC = ACC = Store | VarB" atom le interrup VarA ACC + VarB result int | nic: ts (INTM = 1 o VarC | L) | |

EINT

EINT

ESTOP0

Emulation Stop 0

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|--|---------------------------------|------------------|---------------|--|
| ESTOP0 | | 0111 0110 0010 0101 | Х | - | 1 | |
| Operands | None | | | | | |
| Description | Emulation Stop 0 | | | | | |
| | This instruction is a software breakpoint. | vailable for emulation purpo | oses. It is use | ed to cr | eate a | |
| | When an emulator is connected to the C28x and emulation is enabled, this instruction causes the C28x to halt, regardless of the state of the DBGM bit in status register ST1. In addition, ESTOP0 does not increment the PC. | | | | | |
| | When an emulator is emulation, the EST instruction. It simply | r is not connected or when a debug program has disable STOP0 instruction is treated the same way as a NOI ply advances the PC to the next instruction. | | | | |
| Flags and Modes | None | | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. | |

ESTOP1

Emulation Stop 1

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|--|--|---------------------|-----------------|--|
| ESTOP1 | | 0111 0110 0010 0100 | Х | - | 1 | |
| Operands | None | | | | | |
| Description | Emulation Stop 1 | | | | | |
| | This instruction is av embedded software | vailable for emulation purpo breakpoint. | ses. It is use | d to cre | ate an | |
| | When an emulator is connected to the C28x and emulation is enabled, this instruction causes the C28x to halt, regardless of the state of the DBGM bit in status register ST1. Before halting the processor, ESTOP1 increments the PC so that it points to the instruction following the ESTOP1. | | | | | |
| | When an emulator is emulation, the EST instruction. It simply | not connected or when a d OP0 instruction is treated advances the PC to the ne | ebug program the same wa xt instruction. | n has di ay as a | sabled a NOP | |
| Flags and Modes | None | | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. | |

FFC XAR7,22bit

Fast Function Call

| S | | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------|--|--|---------------------------------|---------------------|---------------|
| FFC XAR7,22t | bit | | 0000 0000 11CC CCCC CCCC CCCC CCCC CCCC | Х | - | 4 |
| Operands | XAR7 22bit | Auxiliary register XA 22-bit program-addr | R7 ess (0x00 0000 to 0x3F FF | FF range) | | |
| Description | | Fast function call. The the 22-bit immediate XAR7 (21:0) = PC + XAR7 (31:22) = 0; | e return PC value is stored e destination address is load 2; | into the XAR ded into the P | 7 regist C: | er and |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | ; Fast FFC | function call of XAR7,FuncA | FuncA: ; Call FuncA, retu | rn address | in XAI | 27 |
| | FuncA: | | ; Function A: | | | |
| | LB | *XAR7 | ; Return: branch t | o address i | n XAR' | 7 |
FLIP AX

Flip Order of Bits in AX Register

| FLIP AX 0101 0110 01 Operands AX Accumulator high (AH) or accum Description Bit reverse the contents of the sp temp = AX; DY (bit 0) | .11 000A ulator low (Al pecified AX re | 1 _) register gister (AH or A | \L): | 1 | | |
|--|--|---|-----------------|---|--|--|
| OperandsAXAccumulator high (AH) or accumDescriptionBit reverse the contents of the sptemp = AX;DY (bit = 0) | ulator low (Al | _) register gister (AH or A | L): | | | |
| Description Bit reverse the contents of the sp temp = AX; | ecified AX re | gister (AH or A | NL): | | | |
| temp = AX; | | | | | | |
| AX (bit 0) = temp(bit 15); AX (bit 1) = temp(bit 14); AX (bit 14) = temp(bit 1); AX (bit 15) = temp(bit 0); Elege and N After the operation if bit 15 of AX | is 1 than the | <pre>temp = AX; AX(bit 0) = temp(bit 15); AX(bit 1) = temp(bit 14); AX(bit 14) = temp(bit 1); AX(bit 15) = temp(bit 0);</pre> | | | | |
| Modes Alter the operation, if bit 15 of AX wise it is cleared. | wise it is cleared. | | | | | |
| Z After the operation, if AX is 0, the | After the operation, if AX is 0, then the Z bit is set, otherwise it is cleared. | | | | | |
| Repeat This instruction is not repeata instruction, it resets the repeat control | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example; Flip the contents of 32-bit vari MOVMOVAH,@VarA+0; LoadMOVAL,@VarA+1; LoadFLIPAL; FlipFLIPAH; FlipFLIPAH; Flip | able VarA: AH with lo AL with hi contents o contents o | w 16 bits of gh 16 bits c f AL f AH | VarA of VarA | 4 | | |

IACK #16bit

Interrupt Acknowledge

| SYNTAX O | PTIONS | OPCODE | OB.IMODE | RPT | CYC |
|--------------------|--|---|---------------------------------|------------------|----------|
| IACK #16bit | | 0111 0110 0011 1111 CCCC CCCC CCCC CCCC | X | - | 1 |
| Operands #16bit | 16-bit constant immediate value (0x0000 to 0xFFFF range) | | | | |
| Description | Acknowledge an interrupt by outputting the specified 16-bit constant on the low 16 bits of the data bus. Certain peripherals will provide the capability to capture this value to provide low-cost trace. See the data sheet for details for your device. data bus(15:0) = 16bit; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the only o | PPT nce. |

Put Processor in Idle Mode

| SYNTAX OPT | TIONS | OPCODE | OBJMODE | RPT | CYC | |
|-------------|--|---|---|---|--|--|
| IDLE | | 0111 0110 0010 0001 | Х | - | 5 | |
| Operands | None | | | | | |
| Description | Put the processor into idle mode and wait for enabled or nonmast terrupt. Devices using the 28x CPU may use the IDLE instruction bination with external logic to achieve different low-power modes. device-specific datasheets for more detail. The idle instruction can following sequence of events: 1) The pipeline is flushed. 2) All outstanding memory cycles are completed. | | | | | |
| | Clocks to the CPU are stopped after the entire instruction but placing the device in the idle state. In the idle state, CLKQ clock output from the CPU) and all clocks to blocks outside (including the emulation block) continue to operate as long a (the clock input to the CPU) is driven. The PC continues to address of the IDLE instruction; the PC is not incremented b CPU enters the idle state. The IDLE output CPU signal is activated (driven high). The device waits for an enabled or nonmaskable hardware If such an interrupt occurs, the IDLESTAT bit is cleared, t incremented by 1, and the device exits the idle state. | | | | r is full, JT (the e CPU CLKIN old the ore the | |
| | | | | | terrupt. PC is | |
| | If the interrupt is m ter (IER). Howeve of the interrupt glo | naskable, it must be enabled er, the device exits the idle s obal mask bit (INTM) of stat | in the interrup tate regardles us register S [−] | t enable s of the F1. | ə regis- ə value | |
| | After the device exits the idle mode, the CPU must respond to the request. If the interrupt can be disabled by the INTM bit in status ST1, the next event depends on INTM: If (INTM = 0), then the interrupt is enabled, and the CPU exe corresponding interrupt service routine. On return from the execution begins at the instruction following the IDLE instruction If (INTM = 1), then the interrupt is blocked and program e continues at the instruction immediately following the IDLE. | | | to the ir status r J execu n the int instruct am exe DLE. | iterrupt egister ites the terrupt, ion. ecution | |
| | If the interrupt car sponding interrup begins at the instr | nnot be disabled by INTM, th t service routine. On return f ruction following the IDLE. | ne CPU exect from the interr | utes the upt, exe | corre- | |

IDLE

IDLE

| Flags and Modes | IDLESTAT | Before entering the idle mode, IDLESTAT is set; after exiting the idle mode IDLESTAT is cleared. |
|--------------------|----------|--|
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-----------------------|--|---------|-----|-----|
| IMACL P,loc32,*XAR7 | 0101 0110 0100 1101 1100 0111 LLLL LLLL | 1 | Y | N+2 |
| IMACL P,loc32,*XAR7++ | 0101 0110 0100 1101 1000 0111 LLLL LLLL | 1 | Y | N+2 |

IMACL P,Ioc32,*XAR7/++ Signed 32 X 32-Bit Multiply and Accumulate (Lower Half)

| Operands | P loc32 *XAR7/++ | Product register Addressing mode (see Chapter 5) Note: The @ACC addressing mode cannot be used when the instruction is repeated. No illegal instruction trap will be generated if used (assembler will flag an error). Indirect program-memory addressing using auxiliary register XAR7, can access full 4Mx16 program space range (0x000000 to 0x3FFFFF) | | |
|-------------|------------------------|--|--|--|
| Description | | 32-bit x 32-bit signed multiply and accumulate. First, add the unsigned previous product (stored in the P register), ignoring the product shift mode (PM), to the ACC register. Then, multiply the signed 32-bit content of the location pointed to by the "loc32" addressing mode by the signed 32-bit content of the program-memory location pointed to by the XAR7 register. The product shift mode (PM) then determines which part of the lower 38 bits of the 64-bit result are stored in the P register. If specified, post-increment the XAR7 register by 1: | | |
| | | <pre>ACC = ACC + unsigned P; temp(37:0) = lower_38 bits(signed [loc32] * signed Prog[*XAR7 or XAR7++]); if(PM = +4 shift) P(31:4) = temp(27:0), P(3:0) = 0; if(PM = +1 shift) P(31:1) = temp(30:0), P(0) = 0; if(PM = 0 shift) P(31:0) = temp(31:0); if(PM = -1 shift) P(31:0) = temp(32:1); if(PM = -2 shift) P(31:0) = temp(33:2); if(PM = -3 shift) P(31:0) = temp(34:3); if(PM = -4 shift) P(31:0) = temp(35:4); if(PM = -5 shift) P(31:0) = temp(36:5); if(PM = -6 shift) P(31:0) = temp(37:6);</pre> | | |

On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*XAR7/++" addressing mode can be used to access data space variables that fall within the program space address range. With some addressing mode combinations, you can get conflicting references. In such cases, the C28x will give the "loc16/loc32" field priority on changes to XAR7. For example:

| IMACL | P,*XAR7,*XAR7++ | ; | XAR7 | given | priority |
|-------|-----------------|---|---------|-------|----------|
| IMACL | P,*XAR7++,*XAR7 | ; | *XAR7++ | given | priority |
| IMACL | P,*XAR7,*XAR7++ | ; | *XAR7++ | given | priority |

| Flags and | Z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | |
|-----------|------|--|--|--|--|--|
| Modes | Ν | | | | | |
| | С | If the addition generates a carry, C is set; otherwise C is cleared. | | | | |
| | V | If an overflow occurs, V is set; otherwise V is not affected. | | | | |
| | OVCU | The overflow counter is incremented when the addition operation gener- ates an unsigned carry. The OVM mode does not affect the OVCU counter. | | | | |
| | РМ | The value in the PM bits sets the shift mode that determines which portion of the lower 38 bits of the 64-bit results are stored in the P register. | | | | |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed N+1 times. The state of the Z, N, C and OVC flags will reflect the final result in the ACC. The V flag will be set if an intermediate overflow occurs in the ACC. | | | | |

```
; Calculate sum of product using 32-bit multiply and retain
Example
           ; 64-bit result:
           ; int32 X[N]; // Data information
            ; int32 C[N]; \ // Coefficient information (located in
                           // low 4M)
           ; int64 sum = 0;
            ; for(i=0; i < N; i++)
            ; sum = sum + (X[i] * C[i]) >> 5;
            ; Calculate low 32 bits:
             MOVL
                   XAR2,#X
                                        ; XAR2 = pointer to X
                   XAR7,#C
                                        ; XAR7 = pointer to C
             MOVL
             SPM
                    -5
                                        ; Set product shift to ">> 5"
             ZAPA
                                        ; Zero ACC, P, OVCU
             RPT
                    #(N-1)
                                        ; Repeat next instruction N times
                                       ; OVCU:ACC = OVCU:ACC + P,
            ||IMACL P,*XAR2++,*XAR7++
                                        ; P = (X[i] * C[i]) << 5,
                                        ; i++
             ADDUL ACC,@P
                                        ; OVCU:ACC = OVCU:ACC + P
                                         ; Store low 32 bits result into sum
             MOVL
                   @sum+0,ACC
            ; Calculate high 32 bits:
             MOVU
                   @AL,OVC
                                       ; ACC = OVCU (carry count)
                   AH,#0
             MOVB
             MPYB
                   P,T,#0
                                        ; P = 0
             MOVL
                   XAR2,#X
                                        ; XAR2 = pointer to X
             MOVL
                   XAR7,#C
                                        ; XAR7 = pointer to C
             RPT
                    #(N-1)
                                       ; Repeat next instruction N times
                                       ; ACC = ACC + P >> 5,
            | | QMACL P, *XAR2++, *XAR7++
                                        ; P = (X[i] * C[i]) >> 32,
                                        ; i++
             ADDL
                   ACC,P << PM
                                        ; ACC = ACC + P >> 5
             MOVL
                    @sum+2,ACC
                                        ; Store high 32 bits result into sum
```

IMPYAL P,XT,Ioc32

Signed 32-Bit Multiply (Lower Half) and Add Previous P

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-------------------|--|---------|-----|-----|
| IMPYAL P,XT,loc32 | 0101 0110 0100 1100 0000 0000 LLLL LLLL | 1 | - | 1 |

| Operands | Р | Product register |
|----------|-------|---------------------------------|
| | ХТ | Multiplicand register |
| | loc32 | Addressing mode (see Chapter 5) |

Description Add the unsigned content of the P register, ignoring the product shift mode (PM), to the ACC register. Multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode. The product shift mode (PM) then determines which part of the lower 38 bits of the 64-bit result are stored in the P register:

```
ACC = ACC + unsigned P;
temp(37:0) = lower_38 bits(signed XT * signed [loc32]);
if (PM = +4 \text{ shift})
   P(31:4) = temp(27:0), P(3:0) = 0;
if ( PM = +1 shift )
   P(31:1) = temp(30:0), P(0) = 0;
if (PM = 0 \text{ shift})
   P(31:0) = temp(31:0);
if ( PM = -1 shift )
   P(31:0) = temp(32:1);
if ( PM = -2 shift )
   P(31:0) = temp(33:2);
if ( PM = -3 shift )
   P(31:0) = temp(34:3);
if( PM = -4 shift )
   P(31:0) = temp(35:4);
if ( PM = -5 shift )
   P(31:0) = temp(36:5);
if ( PM = -6 shift )
  P(31:0) = temp(37:6);
```

| Flags and | Z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | |
|-----------|----|---|--|--|--|--|
| Modes | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | |
| | С | If the addition generates a carry, C is set; otherwise C is cleared. | | | | |
| v ovcu | | If an overflow occurs, V is set; otherwise V is not affected. | | | | |
| | | The overflow counter is incremented when the addition operation generates an unsigned carry. The OVM mode does not affect the OVCU counter. | | | | |
| | РМ | The value in the PM bits sets the shift mode that determines which portion of the lower 38 bits of the 64-bit results are stored in the P register. | | | | |

| Repeat | | This instruction is not instruction, it resets the | re re | epeatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. |
|---------|--------------------|--|----------|--|
| Example | ; Calc: ; Y64 : | ulate signed result: = (X0*C0 + X1*C1 + X2; | *C: | 2) >> 2 |
| | SPM | -2 | : | Set product shift mode to ">> 2" |
| | ZAPA | | ; | Zero ACC, P, OVCU |
| | MOVL | XT,@X0 | ; | XT = X0 |
| | IMPYL | P,XT,@C0 | ; | P = low 32 bits of (X0*C0 << 2) |
| | MOVL | XT,@X1 | ; | XT = X1 |
| | IMPYAL | P,XT,@C1 | ; | OVCU:ACC = OVCU:ACC + P, |
| | | | ; | P = low 32 bits of (X1*C1 << 2) |
| | MOVL | XT,@X2 | ; | XT = X2 |
| | IMPYAL | P,XT,@C2 | ; ; | OVCU:ACC = OVCU:ACC + P, P = low 32 bits of (X2*C2 << 2) |
| | ADDUL | ACC,@P | ; | OVCU:ACC = OVCU:ACC + P |
| | MOVL | @Y64+0,ACC | ; | Store low 32-bit result into Y64 |
| | MOVU | @AL,OVC | ; | ACC = OVCU (carry count) |
| | MOVB | AH,#0 | | |
| | QMPYL | P,XT,@C2 | ; | P = high 32 bits of (X2*C2) |
| | MOVL | XT,@X1 | ; | XT = X1 |
| | QMPYAL | P,XT,@C1 | ; | ACC = ACC + P >> 2, |
| | | | ; | P = high 32 bits of (X1*C1) |
| | MOVL | XT,@X0 | ; | XT = X0 |
| | QMPYAL | P,XT,@C0 | ; | ACC = ACC + P >> 2, |
| | | | ; | P = high 32 bits of (X0*C0) |
| | ADDL | ACC,P << PM | ; | ACC = ACC + P >> 2 |
| | MOVL | @Y64+2,ACC | ; | Store high 32-bit result into Y64 |

IMPYL ACC, XT, loc32

Signed 32 X 32-Bit Multiply (Lower Half)

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--------------------------|--|--|---------|-----|-----|--|--|--|
| IMPYL ACC,X | T,loc32 | | 0101 0110 0100 0100 0000 0000 LLLL LLLL | 1 | - | 2 | | | |
| Operands | ACC XT loc32 | Accumulator register Multiplicand register Addressing mode (see Chapter 5) | | | | | | | |
| Description | | Multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode and store the lower 32 bits of the 64-bit result in the ACC register: ACC = signed XT * signed [loc32]; | | | | | | | |
| Flags and Modes | Z N | After the operation, the After the operation, the After the operation, the operation, the operation, the After the A | After the operation, the Z flag is set if the ACC value is zero, else Z is cleared. After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Calcu MOVL IMPYL | late result: Y32 = M32*X32 + B32 KT,@M32 ; XT = M32 ACC,XT,@X32 ; ACC = low 32 bits of (M32*X32) | | | | | | | |

; ACC = ACC + B32 ; Store result into Y32

ADDL ACC,@B32

MOVL @Y32,ACC

IMPYL P,XT,Ioc32

Signed 32 X 32-Bit Multiply (Lower Half)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| IMPYL P,XT,loc32 | 0101 0110 0000 0101 | 1 | - | 1 |
| | 0000 0000 ГГГГ ГГГГ | | | |

| . . | _ | |
|--------------------|---------|---|
| Operands | Р УТ | Product register Multiplicand register |
| | loc32 | Addressing mode (see Chapter 5) |
| | | 5 (1) |
| Description | | Multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode. The product shift mode (PM) then determines which part of the lower 38 bits of the 64-bit result gets stored in the P register as shown in the diagram below: |
| | | <pre>temp(37:0) = lower_38 bits(signed XT * signed [loc32]); if(PM = +4 shift) P(31:4) = temp(27:0), P(3:0) = 0; if(PM = +1 shift) P(31:1) = temp(30:0), P(0) = 0; if(PM = 0 shift) P(31:0) = temp(31:0); if(PM = -1 shift) P(31:0) = temp(32:1); if(PM = -2 shift) P(31:0) = temp(33:2); if(PM = -3 shift) P(31:0) = temp(34:3); if(PM = -4 shift) P(31:0) = temp(35:4); if(PM = -5 shift) P(31:0) = temp(36:5); if(PM = -6 shift) P(31:0) = temp(37:6);</pre> |
| Flags and Modes | PM | The value in the PM bits sets the shift mode that determines which portion of the lower 38 bits of the 64-bit results are stored in the P register. |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |
| Example | ; Calc | ulate signed result: Y64 = M32*X32 |

| Example | ; Calc | ulate signed result: | ¥6 | 4 = | 143 | 2*X32 | | | | |
|---------|--------|----------------------|----|-----|-----|-------|------|--------|-----|-----------|
| | MOVL | XT,@M32 | ; | XT | = | M32 | | | | |
| | IMPYL | P,XT,@X32 | ; | Ρ | = | low | 32 | bits | of | (M32*X32) |
| | QMPYL | ACC,XT,@X32 | ; | ACC | = | high | 32 | bits | of | (M32*X32) |
| | MOVL | @Y64+0,P | ; | Sto | re | resu | lt : | into ' | Y64 | |
| | MOVL | @Y64+2,ACC | | | | | | | | |
| | | | | | | | | | | |

IMPYSL P,XT,Ioc32

Signed 32-Bit Multiply (Low Half) and Subtract P

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-------------------|--|---------|-----|-----|
| IMPYSL P,XT,loc32 | 0101 0110 0100 0011 0000 0000 LLLL LLLL | 1 | - | 1 |

| Operands | Ρ | Product register |
|----------|---|------------------|
|----------|---|------------------|

XT Multiplicand register

loc32 Addressing mode (see Chapter 5)

Description

Subtract the unsigned content of the P register, ignoring the product shift mode (PM), from the ACC register. Multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode. The product shift mode (PM) then determines which part of the lower 38 bits of the 64-bit result are stored in the P register:

| ACC = ACC - unsigned P; | | | | | | |
|--------------------------------|----------|----|----|---|--------|-----------|
| $temp(37:0) = lower_{38} bit$ | ts(signe | ed | XТ | * | signed | [loc32]); |
| if ($PM = +4 \text{ shift}$) | | | | | | |
| P(31:4) = temp(27:0), | P(3:0) | = | 0; | | | |
| if ($PM = +1$ shift) | | | | | | |
| P(31:1) = temp(30:0), | P(0) | = | 0; | | | |
| if(PM = 0 shift) | | | | | | |
| P(31:0) = temp(31:0); | | | | | | |
| if ($PM = -1$ shift) | | | | | | |
| P(31:0) = temp(32:1); | | | | | | |
| if ($PM = -2$ shift) | | | | | | |
| P(31:0) = temp(33:2); | | | | | | |
| if ($PM = -3$ shift) | | | | | | |
| P(31:0) = temp(34:3); | | | | | | |
| if ($PM = -4$ shift) | | | | | | |
| P(31:0) = temp(35:4); | | | | | | |
| if ($PM = -5 \text{ shift}$) | | | | | | |
| P(31:0) = temp(36:5); | | | | | | |
| if($PM = -6$ shift) | | | | | | |
| P(31:0) = temp(37:6); | | | | | | |
| | | | | | | |

| Flags and Modes | Z | After the subtraction, the Z flag is set if the ACC value is zero, else Z is cleared. |
|--------------------|------|--|
| | Ν | After the subtraction, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | С | If the subtraction generates a borrow, C is cleared; otherwise C is set. |
| | V | If an overflow occurs, V is set; otherwise V is not affected. |
| | OVCU | The overflow counter is decremented when the subtraction operation gener- ates an unsigned borrow. The OVM mode does not affect the OVCU counter. |
| | РМ | The value in the PM bits sets the shift mode that determines which portion of the lower 38 bits of the 64-bit results are stored in the P register. |

| Repeat | This instruction, | tion is not repeatable. If this instruction follows the RPT tresets the repeat counter (RPTC) and executes only once. |
|---------|---|---|
| Example | ; Calculate signed ; Y64 = (-X0*C0 - | 1 result: X1*C1 - X2*C2) >> 2 |
| | SPM -2 | ; Set product shift mode to ">> 2" |
| | ZAPA | ; Zero ACC, P, OVCU |
| | MOVL XT,@X0 | ; XT = X0 |
| | IMPYL P,XT,@CO | ; P = low 32 bits of (X0*C0 << 2) |
| | MOVL XT,@X1 | ; XT = X1 |
| | IMPYSL P,XT,@C1 | ; OVCU:ACC = OVCU:ACC - P, |
| | | ; P = low 32 bits of (X1*C1 << 2) |
| | MOVL XT,@X2 | ; XT = X2 |
| | IMPYSL P,XT,@C2 | ; OVCU:ACC = OVCU:ACC - P, |
| | | ; P = low 32 bits of (X2*C2 << 2) |
| | SUBUL ACC,@P | ; OVCU:ACC = OVCU:ACC - P |
| | MOVL @Y64+0,ACC | ; Store low 32-bit result into Y64 |
| | MOVU @AL,OVC | ; ACC = OVCU (borrow count) |
| | MOVB AH,#0 | |
| | NEG ACC | ; Negate borrow |
| | QMPYL P,XT,@C2 | ; $P = high 32 bits of (X2*C2)$ |
| | MOVL XT,@X1 | ; XT = X1 |
| | QMPYSL P,XT,@C1 | ; ACC = ACC - P >> 2, |
| | | ; $P = high 32 bits of (X1*C1)$ |
| | MOVL XT,@X0 | ; XT = X0 |
| | QMPYSL P,XT,@CO | ; ACC = ACC - $P >> 2$, |
| | | ; $P = nign 32$ bits of (XU*CU) |
| | SUBL ACC, P << PI | A : ACC = ACC - P >> 2 |
| | MOVL @Y64+2,ACC | ; Store high 32-bit result into Y64 |

IMPYXUL P,XT,loc32

Signed 32 X Unsigned 32-Bit Multiply (Lower Half)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|---------|-----|-----|
| IMPYXUL P,XT,loc32 | 0101 0110 0110 0101 0000 0000 LLLL LLLL | 1 | - | 1 |

| Operands | P XT loc32 | Product register Multiplicand register Addressing mode (see Chapter 5) |
|--------------------|------------------|---|
| Description | | Multiply the signed 32-bit content of the XT register by the unsigned 32-bit content of the location pointed to by the "loc32" addressing mode. The product shift mode (PM) then determines which part of the lower 38 bits of the 64-bit result are stored in the P register: |
| | | <pre>temp(37:0) = lower_38 bits(signed XT * unsigned [loc32]); if(PM = +4 shift) P(31:4) = temp(27:0), P(3:0) = 0; if(PM = +1 shift) P(31:1) = temp(30:0), P(0) = 0; if(PM = 0 shift) P(31:0) = temp(31:0); if(PM = -1 shift) P(31:0) = temp(32:1); if(PM = -2 shift) P(31:0) = temp(33:2); if(PM = -3 shift) P(31:0) = temp(34:3); if(PM = -4 shift) P(31:0) = temp(35:4); if(PM = -5 shift) P(31:0) = temp(36:5); if(PM = -6 shift) P(31:0) = temp(37:6);</pre> |
| Flags and Modes | РМ | The value in the PM bits sets the shift mode that determines which portion of the lower 38 bits of the 64-bit results are stored in the P register. |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |

| Example | ; Calcul ; Y64 = | late result: Y Y1:Y0, M64 = 1 | 64 M1 | = M :M0, | 64 X | *X64 + B64 64 = X1:X0, B64 = B1:B0 |
|---------|---------------------|----------------------------------|----------|-------------|---------|---------------------------------------|
| | MOVL | XT,@X0 | ; | ХT | = | XO |
| | IMPYL | P,XT,@M0 | ; | Ρ | = | low 32 bits of (uns M0 * uns X0) |
| | MOVL | ACC,@B0 | ; | ACC | = | BO |
| | ADDUL | ACC,@P | ; | ACC | = | ACC + P |
| | MOVL | @Y0,ACC | ; | Stor | ce | result into YO |
| | QMPYUL | P,XT,@M0 | ; | Ρ | = | high 32 bits of (uns M0 * uns X0) |
| | MOVL | XT,@X1 | ; | XТ | = | X1 |
| | MOVL | ACC,@P | ; | ACC | = | P |
| | IMPYXUL | P,XT,@MO | ; | Ρ | = | low 32 bits of (uns M0 * sign X1) |
| | MOVL | XT,@M1 | ; | XТ | = | Ml |
| | ADDCL | ACC,@P | ; | ACC | = | ACC + P + carry |
| | IMPYXUL | P,XT,@X0 | ; | Ρ | = | low 32 bits of (sign M1 * uns X0) |
| | ADDUL | ACC,@P | ; | ACC | = | ACC + P |
| | ADDUL | ACC,@B1 | ; | ACC | = | ACC + B1 |
| | MOVL | @Y1,P | ; | Stor | ce | result into Y1 |

| IN loc16,*(PA) | | | | Input Data From | | | | |
|----------------|--|---|--|---|---|---|--|--|
| ę | SYNTAX C | PTIONS | | OPCODE | OBJMODE | RPT | CYC | |
| IN loc16,*(PA) | | | 1011 CCCC | 0100 LLLL LLLL CCCC CCCC CCCC | 1 | Y | N+2 | |
| Operands | loc16 *(PA) | Addressing mode Immediate I/O sp | Addressing mode (see Chapter 5) Immediate I/O space memory address | | | | | |
| Description | | Load the location of the specified l | pointed t | to by the "loc16" addr on pointed to by "*(P/ | essing mode w 4)": | ith the | conten | |
| | | [loc16] = IOsp | ace[PA] | ; | | | | |
| | | I/O Space is lim interface (XINTF) device, is toggle lower 16 XINTF zeroed. The data Note: I/O space m particular de | ited to 6), the I/O d during address a is read ay not be in vice for de | 4K range (0x0000 t strobe signal (XIS), the operation. The lines (XA[15:0]) and on the lower 16 data mplemented on all C28x of tails. | o 0xFFFF). Or if available on y I/O address af the upper add lines (XD[15:0 levices. See the d | n the e your pa opears lress lir)]). ata shee | externa Inticulat on the nes are t for you | |
| Flags and | Ν | If (loc16 = @AX) | , then aft | er the move AX is te | sted for a nega | tive co | ndition | |
| Modes | z | If (loc16 = @AX), zero flag bit is se | then after then after t if AX = | er the move, AX is te 0, otherwise it is clea | sted for a zero ared. | conditio | n. The | |
| Repeat | | This instruction is it will be executed is post-incremen | repeatal d N+1 tim ted by 1 | ole. If the operation fo nes. When repeated, during each repetitio | bllows a RPT in the "(PA)" I/O s n. | structic space a | on, ther address | |
| Example | ; IOReg ; IOREg ; IOREg ; IOReg ; IOReg ; IOReg ; IOReg ; if(| gA address = 0x0 gB address = 0x0 gC address = 0x0 gA = 0x0000; gB = 0x0400; gC = VarA; IORegC = 0x2000 IORegC = 0x0000; |)300;)301;)302; | | | | | |
| | IORegA IORegB IORegC MOV | .set 0x0300 .set 0x0301 .set 0x0302 @AL,#0 | ; ; ; | Define IORegA ad Define IORegB ad Define IORegC ad AL = 0 | ldress ldress ldress | | | |
| | UOUT MOV | *(IORegA),@AI @AL,#0x0400 | ; | IOspace[IORegA] AL = 0x0400 | = AL | | | |
| | UOUT | *(IORegB),@AI | ; | IOspace[IORegB] | = AL | | | |
| | UU'I' | * (IUREGC), @Va | arA ; | IUSpace[IURegC] | = varA | | | |
| | CMD | @AI, #0x2000 | ; | AL = IOSpace[IO] Set flags on (A) | $x = 0 \times 2000$ | | | |
| | SB | \$10,NEQ | ; | Branch if not ed | qual | | | |

| MOV | @AL,#0 | ; | AL = 0 | | |
|-------|---------------|---|-----------------|---|----|
| UOUT | *(IORegC),@AL | ; | IOspace[IORegC] | = | AL |
| \$10: | | | | | |

INC loc16

Increment by 1

| S | YNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---------------|--|--|----------------|---------|---------|--|--|
| INC loc16 | | | 0000 1010 LLLL LLLL | Х | - | 1 | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | | | |
| Description | | Add 1 to the signe addressing mode: | ed content of the location | pointed to b | y the ' | 'loc16" | | |
| | | [loc16] = [loc16] | + 1; | | | | | |
| Flags and Modes | Ν | After the operation it | f bit 15 of [loc16] 1, set N; o | therwise, clea | ar N. | | | |
| | Z | After the operation it | f [loc16] is zero, set Z; othe | rwise, clear Z | | | | |
| | С | If the addition gener | ates a carry, C is set; other | wise C is clea | red. | | | |
| | v | If an overflow occurs | s, V is set; otherwise V is no | ot affected. | | | | |
| Repeat | | This instruction is instruction, it resets | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; VarA INC | A = VarA + 1; @VarA | ; Increment content | s of VarA | | | | |

Emulate Hardware Interrupt

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|----------------|--|---|--|---|---|---|
| INTR INTx | | | 0000 0000 0001 CCCC | Х | - | 8 |
| INTR DLOGIN | IT | | 0000 0000 0001 CCCC | Х | - | 8 |
| INTR RTOSIN | Т | | 0000 0000 0001 CCCC | Х | - | 8 |
| INTR NMI | | | 0111 0110 0001 0110 | Х | - | 8 |
| INTR EMUINT | - | | 0111 0110 0001 1100 | Х | - | 8 |
| Operands | INTx DLO- GINT RTOSINT NMI EMUINT | Maskable CPU int Maskable CPU da Maskable CPU re Nonmaskable inte Maskable emulati | terrupt vector name, x = 1 t atalogging interrupt al-time operating system in errupt on interrupt | o 14 terrupt | | |
| Description | | Emulate an intern the interrupt servic instruction. The IN register ST1. It is register (IER) or th INTR instruction r interrupts cannot executing (until th | upt. The INTR instruction to e routine that corresponds to ITR instruction is not affected also not affected by enable ne debug interrupt enable re eaches the decode 2 phas be serviced until the IN e interrupt service routine b | ransfers progr o the vector sp ed by the INT bits in the int egister (DBGII e of the pipel ITR instructio pegins). | ram con becified M bit in cerrupt ER). Or ine, ha ine, ha | ntrol to by the status enable nce the rdware nished |
| INTx | | Interrupt | INTx | Interrupt | | |
| where x = | , | Vector | where x = | Vector | | |
| 0 | | RESET | 9 | INT9 | | |
| 1 | | INT1 | 10 | INT10 | | |
| 2 | | INT2 | 11 | INT11 | | |
| 3 | | INT3 | 12 | INT12 | | |
| 4 | | INT4 | 13 | INT13 | | |
| 5 | | INT5 | 14 | INT14 | | |
| 6 | | INT6 | | | | |
| 7 | | INT7 | | | | |
| 8 | | IN I 8 | | | | |

INTR

Part of the operation involves saving pairs of 16-bit CPU registers onto the stack pointed to by the SP register. Each pair of registers is saved in a single 32-bit operation. The register forming the low word of the pair is saved first (to an even address); the register forming the high word of the pair is saved next (to the following odd address). For example, the first value saved is the concatenation of the T register and the status register ST0 (T:ST0). ST0 is saved first, then T.

This instruction should not be used with vectors 1–12 when the peripheral interrupt expansion (PIE) block is enabled.

| | | <pre>if(not the NMI vector) Clear the corresponding IFR bit; Flush the pipeline; temp = PC + 1; Fetch specified vector; SP = SP + 1; [SP] = T:STO; SP = SP + 2; [SP] = AH:AL; SP = SP + 2; [SP] = PH:PL; SP = SP + 2; [SP] = AR1:ARO; SP = SP + 2; [SP] = DP:ST1; SP = SP + 2; [SP] = DBGSTAT:IER; SP = SP + 2; [SP] = temp; Clear corresponding IER bit; INTM = 0; // disable INT1-INT14, DLOGINT, RTOSINT DBGM = 1; // disable debug events EALLOW = 0; // disable access to emulation registers LOOP = 0; // clear loop flag IDLESTAT = 0; //clear idle flag PC = fetched vector;</pre> |
|--------------------|------------------------|--|
| Flags and Modes | DBGM INTM EALLOW | Debug events are disabled by setting the DBGM bit. Setting the INTM bit disables maskable interrupts. EALLOW is cleared to disable access to protected registers. |
| | LOOP | The loop flag is cleared. |
| | IDLE- STAT | The idle flag is cleared. |

Repeat

This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Interrupt Return

| : | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|--|--|---|--|
| IRET | | | 0111 0110 0000 0010 | Х | - | 8 |
| Operands | | None | | | | |
| Description | | Return from an inter other register value operation. The order in which they were s operations. The stac the register restore of | rrupt. The IRET instruction es that were automatical in which the values are rest aved. All values are popped k pointer is not forced to alig operations: | restores the ly saved by ored is opposi I from the stac n to an even a | PC valu an in te to the k using address | ue and terrupt e order g 32-bit during |
| | | SP = SP - 2; PC = [SP]; SP = SP - 2; | | | | |
| | | DBGSTAT:IER = [SH SP = SP - 2; DP:ST1 = [SP]; | ?]; | | | |
| | | SP = SP - 2; AR1:AR0 = [SP]; | | | | |
| | | <pre>SP = SP - 2; PH:PL = [SP]; SP = SP - 2; AH:AL = [SP]; SP = SP - 2; T:ST0 = [SP]; SP = SP - 1;</pre> | | | | |
| | | Note: Interrupts canno | t be serviced until the IRET instruc | ction completes e | execution | |
| Flags and Modes | SXM OVM TC C Z N V PM OVC | The operation restor | es the state of all flags and | modes of the | ST0 re | egister. |
| | INTM | The operation restore register. The followir | es the state of the specified ng bits are not affected: LO | flags and mod OP, IDLESTA | les of th T, M0M | ne ST1 1MAP |

IRET

IRET

DBGM PAGEO VMAP SPA EAL-LOW AMODE OBJ-MODE XF ARP

Repeat

This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

```
Example
           ; Full interrupt context Save and Restore:
           ; Vector table:
           INTx: .long INTxService ; INTx interrupt vector
             .
             •
           ; Interrupt context save:
           INTxService:
                                         ; ACC, P, T, STO, ST1, DP, ARO,
                                         ; AR1, IER, DPGSTAT registers saved
                                         ; on stack.
                                         ; Return PC saved on stack.
                                         ; IER bit corresponding to INTx
                                         ; is disabled.
                                         ; ST1(EALLOW bit = 0).
                                         ; ST1(LOOP bit = 0).
                                         ; ST1(DBGM bit = 1).
                                         ; ST1(INTM bit = 1).
               PUSH AR1H:AR0H
                                         ; Save remaining registers.
                    XAR2
               PUSH
               PUSH XAR3
               PUSH XAR4
               PUSH XAR5
               PUSH
                    XAR6
                     XAR7
               PUSH
               PUSH
                     XТ
           ; Interrupt user code:
               .
               .
           ; Interrupt context restore:
               POP XT
                                       ; Restore registers.
               POP XAR7
               POP XAR6
               POP XAR5
               POP
                     XAR4
               POP
                     XAR3
               POP XAR2
               POP AR1H:AR0H
               IRET
                                         ; Return from interrupt.
```

Long Indirect Branch

| 5 | SYNTAX OPT | IONS | (| OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|-----------------------------------|--|---|---------------------|----------------|
| LB *XAR7 | | | 0111 013 | 10 0010 0000 | Х | _ | 4 |
| Operands | *XAR7 in ce | idirect program-me ess full 4Mx16 prog | mory addı gram spac | ressing using auxi e range (0x00000 | liary register) 00 to 0x3FFFI | KAR7, (=F) | can ac- |
| Description | Le | ong branch indirect | . Load the | PC with the lower | 22 bits of the X | KAR7 r | egister: |
| | P | C = XAR7(21:0) | ; | | | | |
| Flags and Modes | Ν | lone | | | | | |
| Repeat | T in | his instruction is struction, it resets | not repea | atable. If this in: t counter (RPTC) | struction follo and executes | ows the s only c | ∍ RPT ince. |
| Example | ; Branch SwitchTab .long .long | to subroutines le: Switch0 Switch1 XAR2,#SwitchT | in Switc ; ; ; able ; | hTable selecte Switch address Switch0 address Switch1 address XAR2 = pointe: | d by Switch s table: ss ss r to Switch? | value Table | :: |
| | MOVZ MOVL LB | AR0,@Switch XAR7,*+XAR2[A *XAR7 | ; RO] ; ; | AR0 = Switch XAR7 = Switch Indirect brand | index Table[Switch ch using XAN | 1] R7 | |
| | SwitchRet | urn: | | | | | |
| | Switch0: | | ; | Function A: | | | |
| | LB | SwitchReturn | ; | Return: long] | oranch | | |
| | Switch1: | | ; | Function B: | | | |
| | LB | SwitchReturn | ; | Return: long b | oranch | | |

LB 22bit

Long Branch

| ; | SYNTAX OPTI | ONS | (| OPCODE | OBJMODE | RPT | CYC |
|--------------------|------------------------|---|------------------------|-----------------------------------|--------------------------------------|---------------------|----------------|
| LB 22bit | | | 0000 00 CCCC CC | 00 01CC CCCC CC CCCC CCCC | Х | - | 4 |
| Operands | 22bit 22 | 2-bit program-addr | ess (0x00 | 00000 to 0x3FF | FFF range) | | |
| Description | Lo | ong branch. Load t | he PC wit | th the selected | 22-bit program | address | 6: |
| | PC | C = 22bit; | | | | | |
| Flags and Modes | No | one | | | | | |
| Repeat | Th ins | nis instruction is struction, it resets | not repea the repea | atable. If this t counter (RPT | instruction follo C) and executes | ows the s only o | e RPT ince. |
| Example | ; Branch t ; value: | co subroutines : | in Switc | hTable selec | ted by Switch | | |
| | SwitchTab | le: Switch0 | ; | Switch add: | ress table: dress | | |
| | .long | Switch1 | ; | Switch1 ad | dress | | |
| | | | | | | | |
| | MOVL Table | XAR2,#Switcl | n- ; | XAR2 = poin | nter to Swit | chTab | le |
| | MOVZ | AR0,@Switch | ; | ARO = Swi | ch index | | |
| | MOVL LB | XAR7,*+XAR2 *XAR7 | [AR0] ; ; | Indirect b: | cchTable[Swi ranch using | tch] XAR7 | |
| | SwitchRet | urn: | , | | | | |
| | | | | | | | |
| | Switch0: | | ; | Function A | : | | |
| | • | | | | | | |
| | LB | SwitchReturn | ; | Return: lo | ng branch | | |
| | Switch1: | | ; | Function B | : | | |
| | • | | | | | | |
| | LB | SwitchReturn | ; | Return: lo | ng branch | | |

LC *XAR7

Long Indirect Call

| S | | ONS | 0 | PCODE | OBJMODE | RPT | CYC | |
|--------------------|--|---|-----------------------------|--|--|---------------------|----------------|--|
| LC *XAR7 | | | 0111 013 | LO 0000 0100 | Х | - | 4 | |
| | | | 8 | | | | | |
| Operands | *XAR7 inc | lirect program-me ss full 4Mx16 prog | mory addr gram spac | essing using aux e range (0x0000 | iliary register) 00 to 0x3FFFI | KAR7, (FF) | can ac- | |
| Description | Inc po ad te [S SP [S PC No | <pre>Indirect long call. The return PC value is pushed onto the software stack, pointed to by SP register, in two 16-bit operations. Next, the destination address stored in the XAR7 register is loaded into the PC: temp(21:0) = PC + 1; [SP] = temp(15:0); SP = SP + 1; [SP] = temp(21:16); SP = SP + 1; PC = XAR7(21:0); Note: For more efficient function calls when operating with OBJMODE = 1, use the LCR and LRETR instructions instead of the LC and LRET instructions.</pre> | | | | | | |
| Flags and Modes | Nc | one | | | | | | |
| Repeat | Th ins | is instruction is struction, it resets | not repeation the repeation | atable. If this in counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. | |
| Example | ; Call to SwitchTabl .long .long | subroutines in e: Switch0 Switch1 | SwitchT; ; ; ; | able selected Switch addres Switch0 addre Switch1 addre | by Switch va s table: ss ss | alue: | | |
| | MOVL MOVZ LC | XAR2,#SwitchTa AR0,@Switch XAR7,*+XAR2[A] *XAR7 | able ; ; R0] ; ; | XAR2 = pointe AR0 = Switch XAR7 = Switch Indirect call | r to Switch index Table[Switch using XAR7 | Table | | |
| | Switch0: | | ; | Subroutine 0: | | | | |
| | LRET | | ; | Return | | | | |
| | Switch1: | | ; | Subroutine 1: | | | | |
| | LRET | | ; | Return | | | | |

LC 22bit

Long Call

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|--|--|----------------------------------|---------------------|--------------------|
| LC 22bit | | | 0000 0000 10CC CCCC | Х | - | 4 |
| | | | CCCC CCCC CCCC CCCC | | | |
| Operands | 22bit | 22-bit program-addr | ess (0x00 0000 to 0x3F FF | FF range) | | |
| Description | | Long function call. T pointed to by SP regi destination address | The return PC value is push ister, in two 16-bit operations is loaded onto the PC: | ed onto the so . Next, the im | oftware mediate | stack, e 22-bit |
| | | <pre>temp(21:0) = PC - [SP] = temp(15:0) SP = SP + 1; [SP] = temp(21:16 SP = SP + 1; PC = 22bit;</pre> | + 2;); 5) | | | |
| | | Note: For more efficien LRETR instruction | it function calls when operating with ons instead of the LC and LRET in | n OBJMODE = 1, structions. | use the l | LCR and |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. |
| Example | ; Stand | lard function call | of FuncA: | | | |
| | LC | FuncA | ; Call FuncA, retu | ırn address | on sta | ack |
| | • | | | | | |
| | FuncA: | | ; Function A: | | | |
| | LR | ET | ; Return from addr | ess on stac | :k | |

LCR #22bit

Long Call Using RPC

| | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------------|--|---|---|--|---|
| LCR #22bit | | | 0111 0110 01CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 4 |
| Operands | 22bit | 22-bit program-addi | ress (0x00 0000 to 0x3F FF | FF range) | | |
| Description | | Long call using retu onto the software st Next, the RPC regis immediate destinati | rn PC pointer (RPC). The cu ack, pointed to by SP regist ster is loaded with the retur on address is loaded into th | urrent RPC va er, in two 16-b n address. No le PC: | llue is p bit oper ext, the | oushed ations. 22-bit |
| | | <pre>[SP] = RPC(15:0) SP = SP + 1; [SP] = RPC(21:16 SP = SP + 1; RPC = PC + 2; PC = 22bit;</pre> | ;); | | | |
| | | Note: The LCR and LF LC and LRET o LRETR operation This is the case to be manually s | RETR operations, enable 4 cycle ca perations only enable a 4 cycle cal ons can be nested and can freely rep on interrupts also. Only on a task sv saved and restored. | all and 4 cycle retu I and 8 cycle retu place the LC and vitch operation, do | urn. The s Irn. The L LRET op bes the R | standard LCR and erations. PC need |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | ; RPC o LCF | call of FuncA: R FuncA | ; Call FuncA, retu | ırn address | in RPO | C |
| | FuncA: | | ; Function A: | | | |
| | LRE | ETR | ; RPC return | | | |

| LCR *XARn |
|-----------|
|-----------|

| LCR *XARn | Long Indirect Call Using RPC | | | | | | | | | |
|--------------------|---|---|---|---|--|------------------------------|---------------|--|--|--|
| S | SYNTAX OPT | IONS | (| OPCODE | OBJMODE | RPT | CYC | | | |
| LCR *XARn | | | 0011 11 | 10 0110 ORRR | 1 | - | 4 | | | |
| Operands | * XARn ii) (| indirect program-memory addressing using auxiliary register XAR0 to XAR7, can access full 4Mx16 program space range (0x000000 to 0x3FFFFF) | | | | | | | | |
| Description | L F C T L S S F F F F F | Long indirect call using return PC pointer (RPC). The current RPC value is pushed onto the software stack, pointed to by SP register, in two 16-bit operations. Next, the RPC register is loaded with the return address. Next, the destination address stored in the XARn register is loaded into the PC: [SP] = RPC(15:0); SP = SP + 1; [SP] = RPC(21:16); SP = SP + 1; RPC = PC + 1; PC = XARn(21:0); Note: The LCR and LRETR operations, enable 4 cycle call and 4 cycle return. The standard LC and LRET operations only enable a 4 cycle call and 8 cycle return. The LCR and LRETR operations can be nested and can freely replace the LC and LRET operations. This is the case on interrupts also. Only on a task switch operation, does the RPC need to be manually saved and restored | | | | | | | | |
| Flags and Modes | 1 | lone | | | | | | | | |
| Repeat | ۲ i | This instruction is nstruction, it resets | not repea | atable. If this ins t counter (RPTC) | struction follo | ows the s only o | € RPT nce. | | | |
| Example | ; Call to SwitchTal .long MOVL MOVL LCR Switch0: LRETR Switch1: | <pre>b subroutines in ble: Switch0 Switch1 XAR2,#SwitchTal AR0,@Switch XAR6,*+XAR2[AR0 *XAR6</pre> | SwitchT ; ; ble ; 0] ; ; ; ; | able selected b Switch address Switch0 address Switch1 address XAR2 = pointer AR0 = Switch XAR6 = Switch Indirect RPC of Subroutine 0: RPC Return Subroutine 1: | by Switch va s table: ss to Switch index Table[Switch call using 1 | alue: Table n] XAR6 | | | | |
| | LRETR | | ; | RPC Return | | | | | | |

LOOPNZ loc16,#16bit

Loop While Not Zero

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|---------------------|---------------------|---------|-----|------|
| LOOPNZ loc16,#16bit | 0010 1110 LLLL LLLL | Х | - | 5N+5 |
| | CCCC CCCC CCCC CCCC | | | |

| Operands | loc16 | Addressing mode (see Chapter 5) |
|----------|--------|---|
| | #16bit | 16-bit immediate value (0x0000 to 0xFFFF range) |

while([loc16] & 16bit != 0);

The LOOPNZ instruction uses a bitwise AND operation to compare the value referenced by the "loc16" addressing mode and the 16-bit mask value. The instruction performs this comparison repeatedly for as long as the result of the operation is not 0. The process can be described as follows:

- 1) Set the LOOP bit in status register ST1.
- 2) Generate the address for the value referenced by the "loc16" addressing mode.
- If "loc16" is an indirect-addressing operand, perform any specialized modification to the SP or the specified auxiliary register and/or the ARPn pointer.
- 4) Compare the addressed value with the mask value by using a bitwise AND operation.
- 5) If the result is 0, clear the LOOP bit and increment the PC by 2. If the result is not 0, then return to step 1.

The loop created by steps 1 through 5 can be interrupted by hardware interrupts. When an interrupt occurs, if the LOOPNZ instruction is still active, the return address saved on the stack points to the LOOPNZ instruction. Therefore, upon return from the interrupt the LOOPNZ instruction is fetched again.

While the result of the AND operation is not 0, the LOOPNZ instruction begins again every five cycles in the decode 2 phase of the pipeline. Thus the memory location or register is read once every five cycles. If you use an indirect addressing mode for the "loc16" operand, you can specify an increment or decrement for the pointer (SP or auxiliary register). If you do, the pointer is modified each time in the decode 2 phase of the pipeline. This means that the mask value is compared with a new data-memory value each time.

The LOOPNZ instruction does not flush prefetched instructions from the pipeline. However, when an interrupt occurs, prefetched instructions are flushed.

| | | When any interrupt occurs, the current state of the LOOP bit is saved as ST1 is saved on the stack. The LOOP bit in ST1 is then cleared by the interrupt. The LOOP bit is a passive status bit. The LOOPNZ instruction changes LOOP, but LOOP does not affect the instruction. | | | | | |
|--------------------|------------------------|---|--|--|--|--|--|
| | | You can abort the LOOPNZ instruction within an interrupt service routine. Test the LOOP bit saved on the stack. If it is set, then increment (by 2) the return address on the stack. Upon return from the interrupt, this incremented address is loaded into the PC and the instruction following the LOOPNZ is executed. | | | | | |
| Flags and Modes | Ν | If bit 15 of the result of the AND operation is 1, set N; otherwise, clear N. | | | | | |
| | Z | If the result of the AND operation is 0, set Z; otherwise, clear Z. | | | | | |
| | LOOP | LOOP is repeatedly set while the result of the AND operation is not 0. LOOP is cleared when the result is 0. If an interrupt occurs before the LOOPNZ instruction enters the decode 2 phase of the pipeline, the instruction is flushed from the pipeline and, thus, does not affect the LOOP bit. | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; Wait LOOPN MOV | until bit 3 in RegA is cleared before writing to RegB: NZ @RegA,#0x0004 ; Loop while (RegA AND 0x0004 != 0) @RegB,#0x8000 ; RegB = 0x8000 | | | | | |

LOOPZ loc16,#16bit

Loop While Zero

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|---------|-----|------|
| LOOPZ loc16,#16bit | 0010 1100 LLLL LLLL CCCC CCCC CCCC CCCC | Х | - | 5N+5 |

Operands loc16 Addressing mode (see Chapter 5) #16bit 16-bit immediate value (0x0000 to 0xFFFF range)

Description Loop while zero.

while([loc16] & 16bit = 0);

The LOOPZ instruction uses a bitwise AND operation to compare the value referenced by the "loc16" addressing mode and the 16-bit mask value. The instruction performs this comparison repeatedly for as long as the result of the operation is 0. The process can be described as follows:

- 1) Set the LOOP bit in status register ST1.
- Generate the address for the value referenced by the "loc16" addressing mode.
- If "loc16" is an indirect-addressing operand, perform any specialized modification to the SP or the specified auxiliary register and/or the ARPn pointer.
- 4) Compare the addressed value with the mask value by using a bitwise AND operation.
- 5) If the result is not 0, clear the LOOP bit and increment the PC by 2. If the result is 0, then return to step 1.

The loop created by steps 1 through 5 can be interrupted by hardware interrupts. When an interrupt occurs, if the LOOPZ instruction is still active, the return address saved on the stack points to the LOOPZ instruction. Therefore, upon return from the interrupt the LOOPZ instruction is fetched again.

While the result of the AND operation is 0, the LOOPZ instruction begins again every five cycles in the decode 2 phase of the pipeline. Thus the memory location or register is read once every five cycles. If you use an indirect addressing mode for the "loc16" operand, you can specify an increment or decrement for the pointer (SP or auxiliary register). If you do, the pointer is modified each time in the decode 2 phase of the pipeline. This means that the mask value is compared with a new data-memory value each time.

The LOOPZ instruction does not flush prefetched instructions fr4om the pipeline. However, when an interrupt occurs, prefetched instructions are flushed.

| | | When any interrupt occurs, the current state of the LOOP bit is saved as ST1 is saved on the stack. The LOOP bit in ST1 is then cleared by the interrupt. The LOOP bit is a passive status bit. The LOOPZ instruction changes LOOP, but LOOP does not affect the instruction. | | | | | |
|-----------|------------------------|---|--|--|--|--|--|
| | | You can abort the LOOPZ instruction within an interrupt service routine. Test the LOOP bit saved on the stack. If it is set, then increment (by 2) the return address on the stack. Upon return from the interrupt, this incremented address is loaded into the PC and the instruction following the LOOPZ is executed. | | | | | |
| Flags and | Ν | If bit 15 of the result of the AND operation is 1, set N; otherwise, clear N. If the result of the AND operation is 0, set Z; otherwise, clear Z. | | | | | |
| Modes | Z | | | | | | |
| | LOOP | LOOP is repeatedly set while the result of the AND operation is 0. LOOP is cleared when the result is not 0. If an interrupt occurs before the LOOPZ instruction enters the decode 2 phase of the pipeline, the instruction is flushed from the pipeline and, thus, does not affect the LOOP bit. | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; Wait LOOPZ MOV | until bit 3 in RegA is set before writing to RegB: @RegA,#0x0004 ; Loop while (RegA AND 0x0004 = 0) @RegB,#0x8000 ; RegB = 0x8000 | | | | | |

LPADDR

Set the AMODE Bit

| SYNTAX (| OPTIONS | | OPCODE | OBJMODE | RPT | CYC | | | |
|--|--|-------------------------------------|---|---|------------------------------------|-------------------------|--|--|--|
| LPADDR | | 0101 | 0110 0001 1110 | Х | - | 1 | | | |
| Note: LPADDR is an alias | for the SETC AMODE Oper | ation. | | | | | | | |
| Operands | None | | | | | | | | |
| Description Set the AMODE status bit, putting the device in C2xLP comparadore addressing mode (see Chapter 5). Note: This instruction does not flush the pipeline. | | | | | | | | | |
| Flags and AMODE Modes | The AMODE bit is s | et. | | | | | | | |
| Repeat | This instruction is instruction, it resets | not re the re | epeatable. If this ins peat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. | | | |
| Example ; Exect L L L L A S C | ute the operation PADDR lp_amode DP #VarA ACL VarA DDS VarB ACL VarC 28ADDR | "VarC ; ; ; ; ; ; | = VarA + VarB" wi Full C2xLP addres Tell assembler we Initialize DP (lo ACC = VarA (ACC H ACC = ACC + VarB Store result into Return to C28x ad | citten in C: ss compatible are in C22 ow 64K only) high = 0) (unsigned) o VarC ddress mode | 2xLP sy Le mod KLP mo | ntax: e de | | | |
| | c28_amode | ; | Tell assembler we | ll assembler we are in C28x mode | | | | | |

LRET

Long Return

| SYNT | TAX OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--|--|--------------------------------|-----------|---------|--|--|--|
| LRET | | 0111 0110 0001 0100 | Х | - | 8 | | | |
| Operands | None | | | | | | | |
| Description | Long return. The return PC, in two 16-bit op | urn address is popped, from erations: | the software | stack i | nto the | | | |
| | SP = SP - 1; temp(31:16) = [S] | P]; | | | | | | |
| | SP = SP - 1; | 1 | | | | | | |
| | temp(15:0) = [SP] |]; | | | | | | |
| | $PC = \operatorname{cemp}(21:0);$ | | | | | | | |
| Flags and Modes | None | | | | | | | |
| | Note: For more efficien LRETR instruction | It function calls when operating with ons in place of the LC and LRET in | n OBJMODE = 1, nstructions. | use the I | _CR and | | | |
| Repeat | This instruction is instruction, it resets | This instruction is not repeatable. If this instruction follows the RI instruction, it resets the repeat counter (RPTC) and executes only once | | | | | | |
| Example ; s | Standard function call | of FuncA: | | | | | | |
| • | LC FuncA | ; Call FuncA, retu | rn address | on sta | ack | | | |
| | | | | | | | | |
| | • | | | | | | | |
| Fu | ncA: | ; Function A: | | | | | | |
| | LRET | ; Return from addr | ess on stac | k | | | | |

LRET

LRETE

Long Return and Enable Interrupts

| - | 1 | | | | | | | | |
|--------------------|--------------------------|---|---|-----------------------------|-----------------------------|----------------------------------|---|-----------------------------|--------------------|
| 5 | SYNTAX O | PTIONS | | OPC | CODE | | OBJMODE | RPT | CYC |
| LRETE | | | 0111 | 0110 | 0001 | 0000 | Х | - | 8 |
| Operands | | None | | | | | | | |
| Description | | Long return and ena software stack into the flag (INTM) is cleared SP = SP - 1; temp(31:16) = [SI SP = SP - 1; temp(15:0) = [SP] PC = temp(21:0); INTM = 0; | able int ne PC, ed. This P] ;] ; | errupt: in two s enab | s. The 16-bit les glo | return a operatio obal mas | ddress is pop ns. Next, the <u>c</u> kable interrup | ped, fr lobal ir ots: | om the iterrupt |
| Flags and Modes | INTM | This instruction enal | bles int | terrupt | s by c | learing tl | ne INTM bit. | | |
| Repeat | | This instruction is instruction, it resets | not re the rep | peatal beat co | ble. If ounter | this in: (RPTC) | struction follo and executes | ws the only c | e RPT |
| Example | ; Stand ; enabl LC | lard function call e interrupts on e : FuncA | of Fu xit: ; | uncA. Call | Disak Func | ole inte A, retu | errupts on e | on sta | and ack |
| | FuncA: SE | TC INTM | ; ; | Func Disa | tion ble i | A: nterrup | ts | ŀr | |
| | Lκ | E1E | ; ; | Enab | le in | iterrupt | ess on stat S | κ, | |
LRETR

Long Return Using RPC

| S | SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|---|---|---|
| LRETR | | 0000 0000 0000 0110 | 1 | - | 4 |
| Operands | None | | | | |
| Description | Long return using re RPC register is loader software stack in two PC = RPC; SP = SP - 1; temp(31:16) = [SI SP = SP - 1; temp(15:0) = [SP] RPC = temp(21:0). Note: The LCR and LF LC and LRET operation This is the case of to be manually s | turn PC pointer (RPC). The ed onto the PC. Next, the RP o 16-bit operations: P]; ; RETR operations, enable 4 cycle ca perations only enable a 4 cycle cal ns can be nested and can freely rep on interrupts also. Only on a task sw aved and restored. | return address C register is lo all and 4 cycle retu I and 8 cycle retu place the LC and vitch operation, do | s stored aded fr urn. The s urn. The L LRET op bes the R | t in the om the standard _CR and erations. PC need |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ∕nce. |
| Example | ; RPC call of FuncA: LCR FuncA FuncA: | ; Call FuncA, retu ; Function A: | ırn address | in RPO | 2 |
| | LRETR | ; RPC return | | | |

LSL ACC,#1..16

Logical Shift Left

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSL ACC,#116 | 1111 1111 0011 SHFT | Х | Y | N+1 |

Operands ACC Accumulator register

#1..16 Shift value

Description Perform a logical shift left on the content of the ACC register by the amount specified by the shift value. During the shift, the low order bits of the ACC register are zero filled and the last bit shifted out is stored in the carry flag bit:



- Flags and
ModesNAfter the shift, if bit 31 of ACC is 1 then the negative flag bit is set; otherwise it
is cleared.
 - Z After the shift, if ACC is 0, then the Z bit is set, otherwise it is cleared.
 - **C** The last bit to be shifted out of ACC is stored in C.
- **Repeat** This instruction is repeatable. If the operation follows a RPT instruction, then the LSL instruction will be executed N+1 times. The state of the Z, N, and C flags will reflect the final result.

| Example | ; | Logical | shift left contents | 0 | f VarA by 4: |
|---------|---|---------|---------------------|---|-----------------------------|
| | | MOVL | ACC,@VarA | ; | ACC = VarA |
| | | LSL | ACC,#4 | ; | Logical shift left ACC by 4 |
| | | MOVL | @VarA,ACC | ; | Store result into VarA |

LSL ACC,T

Logical Shift Left by T(3:0)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSL ACC,T | 1111 1111 0101 0000 | Х | - | 1 |

Operands ACC Accumulator register

T Upper 16 bits of the multiplicand (XT) register

Description Perform a logical shift left on the content of the ACC register by the amount specified by the four least significant bits of the T register, T(3:0) = 0...15. Higher order bits are ignored. During the shift, the low order bits of the ACC register are zero filled. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC register:



- Flags and
ModesZAfter the shift, the Z flag is set if the ACC value is zero, else Z is cleared. Even
if the T register specifies a shift of 0, the content of the ACC register is still
tested for the zero condition and Z is affected.
 - N After the shift, the N flag is set if bit 31 of the ACC is 1, else N is cleared. Even if the T register specifies a shift of 0, the content of the ACC register is still tested for the negative condition and N is affected.
 - **C** If (T(3:0) = 0) then C is cleared; otherwise, the last bit shifted out is loaded into the C flag bit.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; | Logical | shift left contents | of VarA by VarB: |
|---------|---|---------|---------------------|------------------------------------|
| | | MOVL | ACC,@VarA | ; ACC = VarA |
| | | MOV | T,@VarB | ; T = VarB (shift value) |
| | | LSL | ACC,T | ; Logical shift left ACC by T(3:0) |
| | | MOVL | @VarA,ACC | ; Store result into VarA |
| | | | | |

LSL AX,#1...16

Logical Shift Left

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSL AX,#116 | 1111 1111 100A SHFT | Х | - | 1 |

Operands AX Accumulator high (AH) or accumulator low (AL) register

#1...16 Shift value

Description Perform a logical shift left on the content of the specified AX register (AH or AL) by the amount given "shift value" field. During the shift, the low order bits of the AX register are zero filled and the last bit to be shifted out is stored in the carry bit flag:



| Flags and | Ν | After the shift, if bit 15 of AX is 1 then the negative flag bit is set; otherwise it is |
|-----------|---|--|
| Modes | | cleared. |

- Z After the shift, if AX is 0, then the Z bit is set, otherwise it is cleared.
- **C** The last bit to be shifted out of AH or AL is stored in C.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Mult: | iply index register a | AR0 | by 2: |
|---------|---------|-----------------------|-----|------------------------------|
| | MOV | AL,@AR0 | ; | Load AL with contents of AR0 |
| | LSL | AL,#1 | ; | Scale result by 1 $(*2)$ |
| | MOV | @AR0,AL | ; | Store result back in AR0 |

LSL AX,T

Logical Shift Left by T(3:0)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSL AX,T | 1111 1111 0110 011A | Х | - | 1 |

- Operands AX Accumulator high (AH) or accumulator low (AL) register
 - T Upper 16 bits of the multiplicand (XT) register

Description Perform a logical shift left on the content of the specified AX register by the amount specified by the four least significant bits of the T register, T(3:0). The contents of higher order bits are ignored. During the shift, the low order bits of the AX register are zero filled. If the T(3:0) register bits specify a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of AX:



- Flags and
ModesNAfter the shift, if bit 15 of AX is 1 then the negative flag bit is set; otherwise it is
cleared. Even if the T(3:0) register bits specify a shift of 0, the value of AH or
AL is still tested for the negative condition and N is affected.
 - Z After the shift, if AX is 0, then the Z bit is set, otherwise it is cleared. Even if the T(3:0) register bits specify a shift of 0, the value of AH or AL is still tested for the zero condition and Z is affected.
 - **C** If T(3:0) specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of AH or AL.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Cal | culate value: | VarC = VarA << VarB; |
|---------|-------|---------------|--------------------------------------|
| | MOV | T,@VarB | ; Load T with contents of VarB |
| | MOV | AL,@VarA | ; Load AL with contents of VarA |
| | LSL | AL,T | ; Scale AL by value in T bits 0 to 3 |
| | MOV | @VarC,AL | ; Store result in VarC |
| | | | |

LSL64 ACC:P,#1..16

Logical Shift Left

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| LSL64 ACC:P,#116 | 0101 0110 1010 SHFT | 1 | - | 1 |

Operands ACC:P Accumulator register (ACC) and product register (P)

#1..16 Shift value

Description Logical shift left the 64-bit combined value of the ACC:P registers by the amount specified in the shift value field. During the shift, the low order bits are zero-filled and the last bit shifted out is stored in the carry bit flag:



- Flags and
ModesNAfter the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and the
N bit is set; otherwise N is cleared.
 - **Z** After the shift, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared.
 - **C** The last bit shifted out of the combined 64-bit value is loaded into the C bit.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Example; Logical shift left the 64-bit Var64 by 10:
MOVL ACC,@Var64+2; Load ACC with high 32 bits of Var64MOVL P,@Var64+0; Load P with low 32 bits of Var64LSL64 ACC:P,#10; Logical shift left ACC:P by 10MOVL @Var64+2,ACC; Store high 32-bit result into Var64MOVL @Var64+0,P; Store low 32-bit result into Var64

LSL64 ACC:P,T

64-Bit Logical Shift Left by T(5:0)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSL64 ACC:P,T | 0101 0110 0101 0010 | 1 | _ | 1 |

Operands ACC:P Accumulator register (ACC) and product register (P)

T Upper 16 bits of the multiplicand register (XT)

Description Logical shift left the 64-bit combined value of the ACC:P registers by the amount specified in the six least significant bits of the T register, T(5:0) = 0...63. Higher order bits are ignored. During the shift, the low order bits are zero-filled. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC:P registers:



- Flags and
ModesNAfter the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and the
N bit is set; otherwise N is cleared.
 - **Z** After the shift, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared.
 - **C** If (T(5:0) = 0) clear C; otherwise, the last bit shifted out of the combined 64-bit value is loaded into the C bit.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Log | ical shift left | the | 64-bit Var64 by contents of Var16: |
|---------|-------|-------------------------|-----|-------------------------------------|
| | MOVL | ACC,@Var64+2 | ; | Load ACC with high 32 bits of Var64 |
| | MOVL | P,@Var64+0 | ; | Load P with low 32 bits of Var64 |
| | MOV | T,@Var16 | ; | Load T with shift value from Var16 |
| | LSL64 | ACC:P,T | ; | Logical shift left ACC:P by T(5:0) |
| | MOVL | <pre>@Var64+2,ACC</pre> | ; | Store high 32-bit result into Var64 |
| | MOVL | @Var64+0,P | ; | Store low 32-bit result into Var64 |

LSLL ACC,T

Logical Shift Left by T (4:0)

| | 000005 | | DDT | 0)/0 |
|----------------|---------------------|---------|-----|------|
| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
| LSLL ACC,T | 0101 0110 0011 1011 | 1 | - | 1 |

| Operands | ACC | Accumulator register |
|----------|-----|---|
| | т | Upper 16 bits of the multiplicand (XT) register |

- T Upper 16 bits of the multiplicand register (XT)

Description Perform a logical shift left on the content of the ACC register by the amount specified by the five least significant bits of the T register, T(4:0) = 0...31. Higher order bits are ignored. During the shift, the low order bits of the ACC register are zero filled. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC register:



- Flags and
ModesZAfter the shift, the Z flag is set if the ACC value is zero, else Z is cleared. Even
if the T register specifies a shift of 0, the content of the ACC register is still
tested for the zero condition and Z is affected.
 - N After the shift, the N flag is set if bit 31 of the ACC is 1, else N is cleared. Even if the T register specifies a shift of 0, the content of the ACC register is still tested for the negative condition and N is affected.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| ; | Logical | shift left contents | of VarA by VarB: |
|---|---------|--|--|
| | MOVL | ACC,@VarA | ; ACC = VarA |
| | MOV | T,@VarB | ; T = VarB (shift value) |
| | LSLL | ACC,T | ; Logical shift left ACC by T(4:0) |
| | MOVL | @VarA,ACC | ; Store result into VarA |
| | ; | ; Logical MOVL MOV LSLL MOVL | ; Logical shift left contents MOVL ACC,@VarA MOV T,@VarB LSLL ACC,T MOVL @VarA,ACC |

LSR AX,#1...16

Logical Shift Right

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSR AX,#116 | 1111 1111 110A SHFT | Х | _ | 1 |

Operands AX Accumulator high (AH) or accumulator low (AL) register

#1...16 Shift value

Description Perform a logical right shift on the content of the specified AX register by the amount given by the "shift value" field. During the shift, the high order bits of the AX register are zero filled and the last bit to be shifted out is stored in the carry flag bit:



Flags andNAfter the shift, if bit 15 of AX is 1 then the negative flag bit is set; otherwise it isModescleared.

Z After the shift, if AX is 0, then the Z bit is set, otherwise it is cleared.

- C The last bit to be shifted out of AH or AL is stored in C.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Example; Divide index register AR0 by 2:MOVAL,@AR0; Load AL with contents of AR0LSRAL,#1; Scale result by 1 (/2)MOV@AR0,AL; Store result back in AR0

LSR AX,T

Logical Shift Right by T(3:0)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSR AX,T | 1111 1111 0110 001A | Х | - | 1 |

| Operands | AX | Accumulator high (AH) or accumulator low (AL) register |
|-------------|----|--|
| | | Upper 16 bits of the multiplicand (XT) register |
| Description | | Perform a logical shift right on the content of the specified AX register (AH or AL) as specified by the four least significant bits of the T register, T(3:0). The contents of higher order bits are ignored. During the shift, the high order bits of the AX register are zero filled If the T(3:0) register bits specify a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of AX: |



| Flags and Modes | Ν | After the shift, if bit 15 of a cleared. Even if the T(3:0 AL is still tested for the r | AX is 1 then the negative flag bit is set; otherwise it is 0) register bits specify a shift of 0, the value of AH or negative condition and N is affected. |
|--------------------|---------|---|--|
| | Z | After the shift, if AX is 0, t T(3:0) register bits speci the zero condition and Z | hen the Z bit is set, otherwise it is cleared. Even if the fy a shift of 0, the value of AH or AL is still tested for Z is affected. |
| | С | If T(3:0) specifies a shift last bit to be shifted out | of 0, then C is cleared; otherwise, C is filled with the of AH or AL. |
| Repeat | | This instruction is not instruction, it resets the | repeatable. If this instruction follows the RPT repeat counter (RPTC) and executes only once. |
| Example | ; Calcu | late un-signed value | : VarC = VarA >> VarB; |
| | MOV | T,@VarB | ; Load T with contents of VarB |
| | MOV | AL,@VarA | ; Load AL with contents of VarA |
| | LSR | AL,T | ; Scale AL by value in T bits 0 to 3 |
| | MOV | @VarC,AL | ; Store result in VarC |

LSR64 ACC:P,#1..16

64-Bit Logical Shift Right

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| LSR64 ACC:P,#116 | 0101 0110 1001 SHFT | 1 | - | 1 |

Operands ACC:P Accumulator register (ACC) and product register (P)

#1..16 Shift value

Description Logical shift right the 64-bit combined value of the ACC:P registers by the amount specified in the shift value field. As the value is shifted, the most significant bits are zero filled and the last bit shifted out is stored in the carry bit flag:



- Flags andNAfter the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and theModesN bit is set; otherwise N is cleared.
 - **Z** After the shift, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared.
 - **C** The last bit shifted out of the combined 64-bit value is loaded into the C bit.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| ; Logical shift right the | 64-bit Var64 by 10: |
|---------------------------|--|
| MOVL ACC,@Var64+2 | ; Load ACC with high 32 bits of Var64 |
| MOVL P,@Var64+0 | ; Load P with low 32 bits of Var64 |
| LSR64 ACC:P,#10 | ; Logical shift right ACC:P by 10 |
| MOVL @Var64+2,ACC | ; Store high 32-bit result into Var64 |
| MOVL @Var64+0,P | ; Store low 32-bit result into Var64 |
| | ; Logical shift right the MOVL ACC,@Var64+2 MOVL P,@Var64+0 LSR64 ACC:P,#10 MOVL @Var64+2,ACC MOVL @Var64+0,P |

LSR64 ACC:P,T

64-Bit Logical Shift Right by T(5:0)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSR64 ACC:P,T | 0101 0110 0101 1011 | 1 | - | 1 |

Operands ACC:P Accumulator register (ACC) and product register (P)

T Upper 16 bits of the multiplicand register (XT)

Description Logical shift right the 64-bit combined value of the ACC:P registers by the amount specified by the six least significant bits of the T register, T(5:0) = 0...63. Higher order bits are ignored. As the value is shifted, the most significant bits are zero filled. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC:P registers:



- Flags andNAfter the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and theModesN bit is set; otherwise N is cleared.
 - **Z** After the shift, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared.
 - **C** If (T(5:0) = 0) clear C; otherwise, the last bit shifted out of the combined 64-bit value is loaded into the C bit.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; Arithmetic shift right | the 64-bit Var64 by contents of Var16: |
|---------|--------------------------|--|
| | MOVL ACC,@Var64+2 | ; Load ACC with high 32 bits of Var64 |
| | MOVL P,@Var64+0 | ; Load P with low 32 bits of Var64 |
| | MOV T,@Var16 | ; Load T with shift value from Var16 |
| | LSR64 ACC:P,T | ; Logical shift right ACC:P by T(5:0) |
| | MOVL @Var64+2,ACC | ; Store high 32-bit result into Var64 |
| | MOVL @Var64+0,P | ; Store low 32-bit result into Var64 |

LSRL ACC,T

Logical Shift Right by T (4:0)

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| LSRL ACC,T | 0101 0110 0010 0010 | 1 | - | 1 |

Operands ACC Accumulator register

T Upper 16 bits of the multiplicand (XT) register

Description Perform an logical shift right on the content of the ACC register as specified by the five least significant bits of the T register, T(4:0) = 0...31. Higher order bits are ignored. During the shift, the high order bits of ACC are zero-filled. If T specifies a shift of 0, then C is cleared; otherwise, C is filled with the last bit to be shifted out of the ACC register:



- Flags and
ModesZAfter the shift, the Z flag is set if the ACC value is zero, else Z is cleared. Even
if the T register specifies a shift of 0, the content of the ACC register is still
tested for the zero condition and Z is affected.
 - N After the shift, the N flag is set if bit 31 of the ACC is 1, else N is cleared. Even if the T register specifies a shift of 0, the content of the ACC register is still tested for the negative condition and N is affected.
 - **C** If (T(4:0) = 0) then C is cleared; otherwise, the last bit shifted out is loaded into the C flag bit.
- **Repeat** This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | ; | Logical | shift right c | contents of VarA by VarB: | |
|---------|---|---------|---------------|-------------------------------------|---|
| | | MOVL | ACC,@VarA | ; ACC = VarA | |
| | | MOV | T,@VarB | ; T = VarB (shift value) | |
| | | LSRL | ACC,T | ; Logical shift right ACC by T(4:0) | ļ |
| | | MOVL | @VarA,ACC | ; Store result into VarA | |

MAC P,loc16,0:pma

Multiply and Accumulate

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-------------------|---------------------|---------|-----|-----|
| MAC P,loc16,0:pma | 0001 0100 LLLL LLLL | Х | - | n+2 |
| | CCCC CCCC CCCC CCCC | | | |

| Operands | Р | Product register | | | | |
|-------------|-------|---|--|--|--|--|
| | loc16 | Addressing mode (see Chapter 5) | | | | |
| | u:pma | space only (0x000000 to 0x00FFFF) | | | | |
| Description | | Add the previous product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. | | | | |
| | | 2) Load the T register with the content of the location pointed to by the "loc16" addressing mode. | | | | |
| | | Multiply the signed 16-bit content of the T register by the signed 16-bit content of the addressed program memory location and store the 32-bit result in the P register: | | | | |
| | | ACC = ACC + P << PM; T = [loc16]; P = signed T * signed Prog[0x00:pma]; | | | | |
| | | The C28x forces the upper 6 bits of the program memory address, specified by the "0:pma" addressing mode, to 0x00 when using this form of the MAC instruction. This limits the program memory address to the low 64K of program address space (0x000000 to 0x00FFFF). On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "0:pma" addressing mode can be used to access data space variables that fall within its address range. | | | | |
| Flags and | Z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | |
| MOUES | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | |
| | С | If the addition generates a carry, C is set; otherwise C is cleared. | | | | |
| | V | If an overflow occurs, V is set; otherwise V is not affected. | | | | |
| | ovc | If overflow mode is disabled; and if the operation generates a positive overflow, then the counter is incremented. If overflow mode is disabled; and if the operation generates a negative overflow, then the counter is decremented. | | | | |

- **OVM** If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFFFFF) or maximum negative (0x8000000) if the operation overflowed.
- **PM** The value in the PM bits sets the shift mode for the output operation from the product register. If the product shift value is positive (logical left shift operation), then the low bits are zero filled. If the product shift value is negative (arithmetic right shift operation), the upper bits are sign extended.
- **Repeat** This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed N+1 times. The state of the Z, N, C and OVC flags will reflect the final result. The V flag will be set if an intermediate overflow occurs. When repeated, the program-memory address is incremented by 1 during each repetition.

```
Example
           ; Calculate sum of product using 16-bit multiply:
           ; int16 X[N] ; Data information
           ; int16 C[N] ; Coefficient information, located in low 64K
           ; sum = 0;
           ; for(i=0; i < N; i++)
            ; sum = sum + (X[i] * C[i]) >> 5;
            MOVL XAR2,#X
                                    ; XAR2 = pointer to X
                                    ; Set product shift to ">> 5"
            SPM -5
            ZAPA
                                    ; Zero ACC, P, OVC
                                   ; Repeat next instruction N times
            RPT #N-1
                                  ; ACC = ACC + P >> 5,
            MAC P,*XAR2++,0:C
                                    ; P = *XAR2++ * *C++
            ADDL ACC,P << PM
                                   ; Perform final accumulate
            MOVL @sum,ACC
                                   ; Store final result into sum
```

MAC P ,loc16,*XAR7/++

Multiply and Accumulate

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-----------------------|--|---------|-----|-----|
| MAC P, loc16, *XAR7 | 0101 0110 0000 0111 1100 0111 LLLL LLLL | 1 | Y | N+2 |
| MAC P, loc16, *XAR7++ | 0101 0110 0000 0111 1000 0111 LLLL LLLL | 1 | Y | N+2 |

| Operands | Р | Product register | | | |
|-------------|---------------------|---|--|--|--|
| | loc16 | Addressing mode (see Chapter 5) | | | |
| | *XAR7 /++ | Indirect program-memory addressing using auxiliary register XAR7, can access full 4M x 16 program space range (0x000000 to 0x3FFFFF) | | | |
| Description | | Use the following steps for this instruction: | | | |
| | | 1) Add the previous product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. | | | |
| | | 2) Load the T register with the content of the location pointed to by the "loc16" addressing mode. | | | |
| | | 3) Multiply the signed 16-bit content of the T register by the signed 16-bit content of the program memory location pointed to by the XAR7 register and store the 32-bit result in the P register. If specified, post-increment the XAR7 register by 1: | | | |
| | | ACC = ACC + P << PM; T = [loc16]; P = signed T * signed Prog[*XAR7 or *XAR7++]; | | | |
| | | On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "XAR7/++" addressing mode can be used to access data space variables that fall within the program space address range. | | | |
| | | With some addressing mode combinations, you can get conflicting references. In such cases, the C28x will give the "loc16/loc32" field priority on changes to XAR7. For example: | | | |
| | | MAC P,*XAR7,*XAR7++ ;XAR7 given priority MAC P,*XAR7++,*XAR7 ; *XAR7++ given priority MAC P,*XAR7,*XAR7++ ; *XAR7++ given priority | | | |

| Flags and | Z | After the addition, the Z fla | ag | is set if the ACC value is zero, else Z is cleared. | | | |
|-----------|--|---|------------------------------|---|--|--|--|
| Modes | Ν | After the addition, the N fl | ag | is set if bit 31 of the ACC is 1, else N is cleared. | | | |
| | С | If the addition generates | a c | carry, C is set; otherwise C is cleared. | | | |
| | v | If an overflow occurs, V is | s s | et; otherwise V is not affected. | | | |
| | OVC | If overflow mode is disa overflow, then the counter the operation generate decremented. | able ris es | ed; and if the operation generates a positive incremented. If overflow mode is disabled; and if a negative overflow, then the counter is | | | |
| | OVM | If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFFFFFF) or maximum negative (0x80000000) if the operation overflowed. | | | | | |
| | РМ | The value in the PM bits s product register. If the operation), then the low negative (arithmetic right | ets pro bi sh | s the shift mode for the output operation from the oduct shift value is positive (logical left shift ts are zero filled. If the product shift value is ift operation), the upper bits are sign extended. | | | |
| Repeat | | This instruction is repeata it will be executed N+1 ti reflect the final result. To occurs. | ble me he | e. If the operation follows a RPT instruction, then es. The state of the Z, N, C and OVC flags will V flag will be set if an intermediate overflow | | | |
| Example | ; Calcu ; int16 ; int16 ; sum = ; for(i ; sum MOVL | <pre>late sum of product us X[N] ; Data informa C[N] ; Coefficient : 0; =0; i < N; i++) = sum + (X[i] * C[i]) XAR2,#X</pre> | sin tic in: >: ; | ng 16-bit multiply: on formation (located in low 4M) > 5; XAR2 = pointer to X | | | |
| | MOVL | XAR7,#C | ; | XAR7 = pointer to C | | | |
| | SPM - | 5 | ; | Set product shift to ">> 5" | | | |
| | ZAPA | | ; | Zero ACC, P, OVC | | | |
| | RPT # | N-1 | ; | Repeat next instruction N times | | | |
| | MAC | P,*XAR2++,*XAR7++ | ; ; | ACC = ACC + P >> 5, P = *XAR2++ * *XAR7++ | | | |
| | ADDL | ACC,P << PM | ; | Perform final accumulate | | | |
| | MOVL | @sum,ACC | ; | Store final result into sum | | | |

MAX AX, loc16

Find the Maximum

| 5 | Ο ΧΑΤΑΥ | PTIONS | OPC | ODE | OBJMODE | RPT | CYC |
|--------------------|---|---|------------------------------------|--|---|------------------------|-----------------|
| MAX AX, loc16 | 6 | | 0101 0110 0000 0000 | 0111 001A LLLL LLLL | 1 | Y | N+1 |
| Operands | AX | Accumulator high (A | AH) or accum | ulator low (AL) | register | | |
| | loc16 | Addressing modes | (see Chapter | 5) | | | |
| Description | | Compare the signed contents of the specified AX register (AH or AL) with the signed content of the location pointed to by the "loc16" addressing mode are load the AX register with the larger of these two values: | | | | | |
| | | if(AX < [loc16]) if(AX >= [loc16] | , AX = [loc), AX = unc | 16]; nanged; | | | |
| Flags and Modes | N | If AX is less then the the negative flag bit | e contents of t will be set; ot | he addressed herwise, it wil | location (AX < I be cleared. | < [loc16 | i]) then |
| | Z | If AX and the content the zero flag bit will | ts of the addre be set; otherv | ssed location a vise, it will be | are equal (AX cleared. | = [loc16 | 3]) then |
| | V | If AX is less then the the overflow flag bit | e contents of t will be set. Tl | he addressed his instruction | location (AX < cannot clear | < [loc16 the V fl | ة]) then ag. |
| Repeat | | If the operation is fo N+1 times. The stat | llows a RPT in e of the N, Z, | nstruction, the and V flags w | instruction wi ill reflect the f | ll be ex inal res | ecuted sult. |
| Example | ; Satu: ; if(Va ; if(Va MOV Al MOV @/ MIN Al NEG Al MAX Al MOV @' | rate VarA as follo arA > 2000) VarA = arA < -2000) VarA L,@VarA AH,#2000 L,@AH H L,@AH VarA,AL | ws: 2000; = -2000; | ; Load AL w; ; Load AH w; ; if(AL > A] ; AH = -2000 ; if(AL < A] ; Store rest | ith content; ith the valu H) AL = AH O H) AL = AH ult into Va: | s of V ue 200 rA | arA O |

| MAXCUL P,loc32 Conditionally Find the Unsigned Maxim | | | | | | kimum | | |
|--|--|---|---|--|---|--------------------------------------|--|--|
| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
| MAXCUL P, loc | 32 | | 0101 0110 0101 0001 0000 0000 LLLL LLLL | 1 | - | 1 | | |
| Operands | Р | Product register | | | | | | |
| | loc32 | Addressing mode (s | ee Chapter 5) | | | | | |
| Description | | Based on the state of contents of the P re- pointed to by the "loo larger of the two nur | f the N and Z flags, conditio gister with the 32-bit, unsig c32" addressing mode and nbers: | nally compare ned content c load the P ree | e the un of the lo gister w | signed ocation vith the | | |
| | | <pre>if((N=1) & (Z=0)) P = [loc32]; if((N=0) & (Z=1) & (P < [loc32])) V=1, P = [loc32]; if((N=0) & (Z=0)) </pre> | | | | | | |
| | | Note: The "P < [loc32" | operation is treated like a 32-bit u | nsigned compare | e. | | | |
| | | This instruction is ty 64-bit maximum func by using a MAXL ins The MAXCUL instru- bits based on the res | pically combined with the lettion. It is assumed that the lettion. It is assumed that the lettruction to compare the upper tion is then used to condition sults of the upper 32-bit conditions. | MAXL instruc N and Z flags per 32 bits of a onally compar- mparison. | tion to will first a 64-bit e the lo | form a be set value. wer 32 | | |
| Flags and | Ν | If $(N = 1 \text{ and } z = 0)$ the | en load P with [loc32]. | | | | | |
| Modes | Z | If $(N = 0 \text{ and } Z = 1) \text{ constant}$ [loc32] and load P with | mpare the unsigned content o the larger of the two. | f the P with the | unsigne | əd | | |
| | | If $(N = 0 \text{ and } Z = 0)$ do | nothing. | | | | | |
| | V | If $(N = 0 AND Z = 1 AN)$ | ND P < [loc32]) then V is set; (| otherwise, V is | unchan | ged. | | |
| Repeat | | This instruction is not r resets the repeat coun | repeatable. If this instruction for ter (RPTC) and executes only | ollows the RPT once. | instruct | ion, it | | |
| Example | ; Satur ; if(Va ; if(Va MOVL P MOVL P MINL P MINCUL SB sa | cate 64-bit Var64 a ar64 > MaxPos64) ar64 < MaxNeg64) ACC,@Var64+2 P,@Var64+0 ACC,@MaxPos64+2 P,@MaxPos64+2 aturate,OV | as follows: Var64 = MaxPos64 Var64 = MaxNeg64 ; Load ACC:P with V ; if(ACC:P > MaxPos | /ar64 364) ACC:P = | = MaxP | 0564 | | |
| | MAXL A | P,@MaxNeg64+2 P,@MaxNeg64+0 | ; 11(ACC:P < MaxNeg | J64) ACC:P = | = MaxN | eg64 | | |

Conditionally Find the Unsigned Maximum

Saturate: MOVL @Var64+2,ACC ; Store result into Var64 MOVL @Var64,P

MAXL ACC, loc32

Find the 32-bit Maximum

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|--|--|---|-------------------------------|-------------------------------|
| MAXL ACC,loc | 32 | | 0101 0110 0110 0001 0000 0000 LLLL LLLL | 1 | Y | N+1 |
| Operands | ACC | Accumulator registe | r | | | |
| | loc32 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Compare the conten "loc32" addressing n two values: | t of the ACC register with th node and load the ACC regi | ne location poi ster with the l | nted to arger o | by the f these |
| | | if(ACC < [loc32]) if(ACC >= [loc32] |), ACC = [loc32];), ACC = unchanged; | | | |
| Flags and Modes | Z | If ACC is equal to the contents of the addressed location (ACC = $[loc32]$), set Z; otherwise, clear Z. | | | | |
| | Ν | If ACC is less then the contents of the addressed location, (ACC < [loc32]), set N; otherwise clear N. The MAXL instruction assumes infinite precision when it determines the sign of the result. For example, consider the subtraction 0x8000 0000 – 0x0000 0001. If the precision were limited to 32 bits, the result would cause an overflow to the positive number 0x7FFF FFFF and N would be cleared. However, because the MAXL instruction assumes infinite precision, it would set N to indicate that 0x8000 0000 – 0x0000 0001 actually results in a pegative number | | | | |
| | С | lf (ACC – [loc32]) ge | enerates a borrow, clear the | C bit; otherw | ise set | C. |
| | V | If ACC is less then the set V. This instruction | he contents of the addresse n cannot clear the V flag. | ed location (A | CC < [l | oc32]), |
| Repeat | | This instruction is rep the MAXL instruction flags will reflect the overflow occurs. | peatable. If the operation foll will be executed N+1 times. final result. The V flag wi | ows a RPT in The state of t Il be set if an | structio he Z, N interm | n, then , and C iediate |
| Example | ; Satu: ; if(Va ; if(Va MOVL MINL MAXL MOVL | rate VarA as follo arA > MaxPos) VarA arA < MaxNeg) VarA ACC,@VarA ACC,@MaxPos ACC,@MaxNeg @VarA,ACC | ws: = MaxPos = MaxNeg ; ACC = VarA ; if(ACC > MaxPos ; if(ACC < MaxNeg ; Store result in | s) ACC = Ma: g) ACC = Ma: nto VarA | xPos xNeg | |

MIN AX, loc16

Find the Minimum

| S | Ο ΧΑΤΛΥ | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-------------------------------|---|--|-------------------------------|---------------------|----------------|--|
| MIN AX, loc16 | | | 0101 0110 0111 010A 0000 0000 LLLL LLLL | 1 | Y | N+1 | |
| Operands | AX | Accumulator high (A | H) or accumulator low (AL) | register | | | |
| | loc16 | Addressing modes (| see Chapter 5) | | | | |
| Description | | Compare the signed content of the specified AX register (AH or AL) with the content of the signed location pointed to by the "loc16" addressing mode and load the AX register with the smaller of these two values: | | | | | |
| | | if(AX > [loc16]), if(AX <= [loc16]) | AX = [loc16]; , AX = unchanged; | | | | |
| Flags and Modes | Ν | If AX is less then the the negative flag bit | contents of the addressed will be set; otherwise, it will | location (AX < be cleared. | < [loc16 | i]) then | |
| | Z | If AX and the content the zero flag bit will | s of the addressed location a be set; otherwise, it will be | are equal (AX) cleared. | = [loc16 | 3]) then | |
| | V | If AX is greater then the contents of the addressed location $(AX > [loc16])$ then the overflow flag bit will be set. This instruction cannot clear the V flag. | | | | | |
| Repeat | | If the operation is fol N+1 times. The state | lows a RPT instruction, the e of the N, Z and V flags wil | instruction wil | l be ex nal resi | ecuted ult. | |
| Example | ; Satur ; if(Va ; if(Va | urate VarA as follows: /arA > 2000) VarA = 2000; /arA < -2000) VarA = -2000; | | | | | |
| | MOV AI | ,@VarA | ; Load AL with conte | ents of VarA | 1 | | |
| | MOV @A MIN AI | ΔΗ,#2000 ,@AH | ; LOAD AH WITH THE V ; $if(AL > AH) AL = A$ | H | | | |
| | NEG AF | I | ; $AH = -2000$ | | | | |
| | MAX AI | ,@AH | ; if $(AL < AH) AL = A$ | Н | | | |
| | ™OV @1 | /arA,AL | ; Store result into | VarA | | | |

MINCUL P, loc32

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|---|---|--|---|--------------------------------------|--|--|
| MINCUL P, loc3 | 32 | | 0101 0110 0101 1001 xxxx xxxx LLLL LLLL | 1 | - | 1 | | |
| Operands | Р | Product register | | | | | | |
| | loc32 | Addressing mode (s | see Chapter 5) | | | | | |
| Description | | Based on the state of contents of the P re pointed to by the "lo smaller of the two n | Based on the state of the N and Z flags, conditionally compare the unsigned contents of the P register with the 32-bit, unsigned content of the location pointed to by the "loc32" addressing mode and load the P register with the smaller of the two numbers: | | | | | |
| | | <pre>if((N = 0) & (Z P = [loc32]; if((N = 0) & (Z V=1, P = [loc3: if((N = 1) & (Z P = unchanged;</pre> | <pre>if((N = 0) & (Z = 0)) P = [loc32]; if((N = 0) & (Z = 1) & (P > [loc32])) V=1, P = [loc32]; if((N = 1) & (Z = 0)) P = unchanged;</pre> | | | | | |
| | | Note: The "p < [loc32] | " operation is treated like a 32-bit ι | unsigned compar | e. | | | |
| | | This instruction is ty 64-bit minimum func by using a MINL ins The MINCUL instruc bits based on the re | ypically combined with the ction. It is assumed that the l struction to compare the upp ction is then used to condition esults of the upper 32-bit con | MINL instruct N and Z flags per 32 bits of a onally compare mparison. | tion to will first a 64-bit e the lo | form a be set value. wer 32 | | |
| Flags and Modes | N Z | If (N = 1 AND Z = 0) If (N = 0 AND Z =1), to [loc32]. If (N = 0 AND Z = 0) |), then load the P register w compare unsigned and load), do nothing. | ith [loc32]. P with the sm | aller P ı | register | | |
| | V | If (N = 0 AND Z = 1 A | AND P < [loc32]) then V is se | t; otherwise, V | is unch | anged. | | |
| Repeat | | This instruction is no instruction, it resets | ot repeatable. If this instruct the repeat counter (RPTC) | tion follows th and executes | e RPT s only c | nce. | | |
| Example | ; Satu ; if(V ; if(V MOVL MOVL | ; Saturate 64-bit Var64 as follows: ; if(Var64 > MaxPos64) Var64 = MaxPos64 ; if(Var64 < MaxNeg64) Var64 = MaxNeg64 MOVL ACC,@Var64+2 ; Load ACC:P with Var64 MOVL P,@Var64+0 | | | | | | |
| | MINCUI | L P,@MaxPos64+0 | , II (ACC: F > MAXPOSE | $A = \frac{1}{2} A = $ | Marti | | | |
| | MAXL MAXCUI | ACC,@MaxNeg64+2 L P,@MaxNeg64+0 | ; if(ACC:P < MaxNeg6 | 04) ACC:P = | MaxNeg | g64 | | |
| | MOVL MOVL | @Var64+2,ACC @Var64+0,P | ; Store result into | Var64 | | | | |

MINL ACC, loc32

Find the 32-bit Minimum

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-----------------|--|---------|-----|-----|
| MINL ACC, loc32 | 0101 0110 0101 0000 0000 0000 LLLL LLLL | 1 | Y | N+1 |

| Operands | ACC | Accumulator register | | | |
|--------------------|-------------------------------|---|--|--|--|
| | loc32 | Addressing mode (see Cha | apter 5) | | |
| Description | | Compare the content of the "loc32" addressing mode a two values: | e ACC register with the location pointed to by the nd load the ACC register with the larger of these | | |
| | | if(ACC <= [loc32]), AC if(ACC > [loc32]), ACC | C = unchanged; C = [loc32]; | | |
| Flags and Modes | Z | If ACC is equal to the conte Z; otherwise clear Z. | nts of the addressed location (ACC = [loc32]), set | | |
| | Ν | If ACC is less then the conset N; otherwise clear N. when it determines the subtraction 0x8000 0000 – bits, the result would cause and N would be cleared. H infinite precision, it would s actually results in a negative | tents of the addressed location, (ACC < [loc32]), The MINL instruction assumes infinite precision sign of the result. For example, consider the 0x0000 0001. If the precision were limited to 32 an overflow to the positive number 0x7FFF FFFF owever, because the MINL instruction assumes et N to indicate that 0x8000 0000 – 0x0000 0001 ve number. | | |
| | С | If (ACC - [loc32]) generate | es a borrow, clear the C bit; otherwise set C. | | |
| | v | If ACC is less then the con set V. This instruction canr | tents of the addressed location (ACC < [loc32]), not clear the V flag. | | |
| Repeat | | This instruction is repeatab the MINL instruction will be flags will reflect the final r overflow occurs. | le. If the operation follows a RPT instruction, then executed N+1 times. The state of the Z, N, and C result. The V flag will be set if an intermediate | | |
| Example | ; Satur ; if(Va ; if(Va | ate VarA as follows: rA > MaxPos) VarA = Maz rA < MaxNeg) VarA = Maz | kPos kNeg | | |
| | MOVL | ACC,@VarA | ; ACC = VarA | | |
| | MINL | ACC,@MaxPos | ; if(ACC > MaxPos) ACC = MaxPos | | |
| | MAXL | ACC,@MaxNeg | ; if(ACC < MaxNeg) ACC = MaxNeg | | |
| | MOVL | @VarA,ACC | ; Store result into VarA | | |

MOV *(0:16bit), loc16

Move Value

| 5 | SYNTAX OPT | IONS | OPC | ODE | OBJMODE | RPT | CYC |
|--------------------|------------|--|--|---|--|--------------------------------|--------------------------------|
| MOV *(0:16bit) |),loc16 | | 1111 0100 1 CCCC CCCC | LLLL LLLL CCCC CCCC | Х | Y | N+2 |
| Operands | *(0:16bit) | Immediate direct space only (0x00 | memory addr 000000 to 0x0 | ress, access lo 0000FFFF) | w 64K range | of data | l |
| | loc16 | Addressing mode | (see Chapte | er 5) | | | |
| Description | | Move the content mode to the mem [0x0000:16bit] | t of the locati ory location s = [loc16] ; | ion pointed to pecified by the | by the "loc16 "0:16bit" cons | 6" addr stant ac | essing Idress: |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is then it will be e data-memory add Only the lower 16 | repeatable. I xecuted N+1 Iress is post- bits of the ad | If the operatior times. Wher incremented b ddress is affec | n follows a RF n repeated, t y 1 during ea ted. | PT instr he "(0: ich rep | uction, 16bit)" etition. |
| | | ; Copy the contents of Array1 to Array2: ; int16 Array1[N]; ; int16 Array2[N]; // Located in low 64K of data space ; for(i=0; i < N; i++) ; Array2[i] = Array1[i]; | | | | | ce |
| Example | | | | | | | |
| | MOVL XAR | 2,#Array1 | ; XA | R2 = pointer | to Arrayl | | |
| | RPT #(N | [-1] | ; Re | peat next ir | struction 1 |] time | S |
| | MOV *(| 0:Array2),*XAR2- | ++ ; Ar ; i+ | ray2[i] = Ar + | rayl[i], | | |

MOV ACC,#16bit<<#0..15

Load Accumulator With Shift

| S | NTAX OF | TIONS | OPCODE | OBJMODE | RPT | CYC | | |
|---------------|--|---|--|----------------|-----------|-------|--|--|
| MOV ACC, loc1 | 6<<#015 | | 1111 1111 0010 SHFT CCCC CCCC CCCC CCCC | Х | - | 1 | | |
| Operands | ACC #16bit #015 | Accumulator register 16-bit immediate constant value Shift value (default is "<< #0" if no value specified) | | | | | | |
| Description | | Load the ACC register with the left shifted contents of the 16-bit immediate value. The shifted value is sign extended if sign extension mode is turned on (SXM = 1) else the shifted value is zero extended (SXM = 0). The lower bits of the shifted value are zero filled: if(SXM = 1) // sign extension mode enabled ACC = S:16bit << shift value; else // sign extension mode disabled ACC = 0:16bit << shift value; | | | | | | |
| Flags and | N | After the load, the | N flag is set if bit 31 of the AC | C is 1, else N | l is clea | ared. | | |
| Modes | z | After the load, the | Z flag is set if the ACC value i | s zero, else Z | Z is clea | ared. | | |
| | SXM | If sign extension mode bit is set; then the 16-bit constant operand sign extended before the load; else, the value will be zero extended | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Calcu SETC SX MOV AC ADD AC | <pre>.late signed value: ACC = -2010 << 10 + VarB << 6; M ; Turn sign extension mode on CC,#-2010 << #10 ; Load ACC with -2010 left shifted by 10 CC,@VarB << #6 ; Add VarB left shifted by 6 to ACC</pre> | | | | | | |

MOV ACC,loc16<<T

Load Accumulator With Shift

| S | YNTAX C | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|---|--|---|---|-------------------------------------|--|--|
| MOV ACC,loc1 | 6 << T | | 0101 0110 0000 0110 0000 0000 LLLL LLLL | 1 | - | 1 | | |
| Operands | ACC loc16 T | Accumulator regist Addressing mode Upper 16 bits of th | Accumulator register Addressing mode (see Chapter 5) Jpper 16 bits of the multiplicand register, XT(31:16) | | | | | |
| Description | | Load the ACC reg pointed to by the "k four least significar order bits are ignor mode is turned of (SXM = 0). The low if (SXM = 1) / ACC = S: [loc1 else // s ACC = 0: [loc1 | Load the ACC register with the left-shifted contents of the 16-bit location pointed to by the "loc16" addressing mode. The shift value is specified by the four least significant bits of the T register, T(3:0) = shift value = 015. Highe order bits are ignored. The shifted value is sign extended if sign extension mode is turned on (SXM = 1) else the shifted value is zero extended (SXM = 0). The lower bits of the shifted value are zero filled: if (SXM = 1) // sign extension mode enabled ACC = S: [loc16] << T(3:0); else // sign extension mode disabled ACC = 0: [loc16] << T(3:0); | | | | | |
| Flags and Modes | N Z SXM | After the load, the After the load, the If sign extension m "loc16" field, will be extended. | N flag is set if bit 31 of the AC Z flag is set if the ACC value i node bit is set; then the 16-bit o sign extended before the load; | C is 1, else N s zero, else z operand, addr ; else the valu | l is clea ː is clea essed e will b | ared. ared. by the le zero | | |
| Repeat | | This instruction is i instruction, it reset | not repeatable. If this instructions the repeat counter (RPTC) a | on follows the and executes | RPT only o | nce. | | |
| Example | ; Calo SETC S MOV 2 MOV 2 MOV 2 ADD 2 | culate signed valu SXM F,@SA ACC,@VarA << T F,@SB ACC,@VarB << T | <pre>e: ACC = (VarA << SB) + ; Turn sign extension ; Load T with shift v ; Load in ACC shifted ; Load T with shift v ; Add to ACC shifted</pre> | (VarB << SB mode on alue in SA contents c alue in SB contents of |) of Var. VarB | A | | |

MOV ACC, loc16<<#0..16

Load Accumulator With Shift

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------------|--|---------|-----|--------|
| MOV ACC,loc16<<#0 | 1000 0101 LLLL LLLL 1110 0000 LLLL LLLL | 1 0 | - | 1 1 |
| MOV ACC, loc16<<#115 | 0101 0110 0000 0011 0000 SHFT LLLL LLLL | 1 | - | 1 |
| | 1110 SHFT LLLL LLLL | 0 | - | 1 |
| MOV ACC, loc16<<#16 | 0010 0101 LLLL LLLL | Х | - | 1 |

| Operands | ACC loc16 #016 | Accumulator register Addressing mode (see Chapter 5) Shift value (default is "<< #0" if no value specified) |
|--------------------|--|---|
| Description | | Load the ACC register with the left shifted contents of the addressed location pointed to by the "loc16" addressing mode. The shifted value is sign extended if sign extension mode is turned on $(SXM = 1)$ else the shifted value is zero extended $(SXM = 0)$. The lower bits of the shifted value are zero filled: |
| | | <pre>if(SXM = 1) // sign extension mode enabled ACC = S:[loc16] << shift value; else // sign extension mode disabled ACC = 0:[loc16] << shift value;</pre> |
| Flags and Modes | Ν | After the load, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | Z | After the load, the Z flag is set if the ACC is zero, else Z is cleared. |
| | SXM | If sign extension mode bit is set; then the 16-bit operand, addressed by the "loc16" field, will be sign extended before the load; else the value will be zero extended. |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |
| Example | ; Calcu SETC SX MOV AC ADD AC | late signed value: ACC = VarA << 10 + VarB << 6; M ; Turn sign extension mode on C,@VarA << #10 ; Load ACC with VarA left shifted by 10 C,@VarB << #6 ; Add VarB left shifted by 6 to ACC |
| | | |

Modes

MOV AR6/7, loc16

Load Auxiliary Register

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|----------------|----------------|--|--|---------|-----|-----|--|--|
| MOV AR6, loc1 | 6 | | 0101 1110 LLLL LLLL | Х | _ | 1 | | |
| MOV AR7, loc1 | 6 | | 0101 1111 LLLL LLLL | Х | - | 1 | | |
| Operands | AR6/7 loc16 | AR6 or AR7, auxilia Addressing mode (| ary registers 'see Chapter 5) | | | | | |
| Description | | Load AR6 or AR7 v 16 bits of XAR6 an AR6/7 = [loc16] ; AR6/7H = unchang | Load AR6 or AR7 with the contents of the 16-bit location and leave the upper 16 bits of XAR6 and XAR7 unchanged: AR6/7 = [loc16]; AR6/7H = unchanged; | | | | | |
| Flags and | | None | | | | | | |

RepeatThis instruction is not repeatable. If this instruction follows the RPT
instruction, it resets the repeat counter (RPTC) and executes only once.

MOV AX, loc16

Load AX

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-------|--|--|--|---------------------------------|-------------------------------|--|
| MOV AX, loc16 | 6 | | 1001 001A LLLL LLLL | Х | - | 1 | |
| Operands | АХ | Accumulator high (A | H) or accumulator low (AL) | register | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | | |
| Description | | Load accumulator l register with the 16 addressing mode, unchanged: | high register (AH) or acc -bit contents of the locatio leaving the other half of | umulator low n pointed to k the accumu | registe by the ' lator re | er (AL) "loc16" egister | |
| | | AX = [loc16]; | | | | | |
| Flags and Modes | N | The load to AX is te this flag is set; other | sted for a negative conditio wise it is cleared. | n. If bit 15 of / | AX is 1, | , then | |
| | Z | The load to AX is tea results in AX = 0, otl | The load to AX is tested for a zero condition. The bit is set if the operation results in $AX = 0$, otherwise it is cleared | | | | |
| Repeat | | This instruction is instruction, it resets | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | |
| Example | MOV | AH, *+XAR0[0] | ; Load AH with the ; of location poin ; AL is unchanged. | 16-bit cont ted to by XA | ents ARO. | | |
| | SB | NotZero,NEQ | ; Branch if conten ; zero. | ts of AH wei | re non | | |

MOV DP, #10bit

Load Data-Page Pointer

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---------------------|--|---|---------|-----|-----|--|--|
| MOV DP, #10bit | 1 | | 1111 10CC CCCC CCCC | X | - | 1 | | |
| Operands | DP #10bit | Data page register 10-bit immediate con | stant value | | | | | |
| Description | | Load the data page i unchanged: | _oad the data page register with a 10-bit constant leaving the upper 6 bits unchanged: | | | | | |
| | | DP(9:0) = 10bit; DP(15:10) = uncha | nged; | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | MOV DP, #VarA | ; Load DP with th ; contains VarA. ; the lower 0x000 ; DP(15:10) is le | e data page that Assumes VarA is in 0 FFC0 of memory. ft unchanged. | | | | | |

MOV IER, loc16

Load the Interrupt-Enable Register

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-------------------------------------|--|--|---------------------------------|-------------------|---------|
| MOV IER, loc16 | ; | | 0010 0011 LLLL LLLL | Х | - | 5 |
| Operands | IER loc16 | Interrupt-enable regi Addressing mode (s | ster ee Chapter 5) | | | |
| Description | | Enable and disable s pointed to by the "loc IER = [loc16]; | selected interrupts by loading c16" addressing mode into | g the content the IER regist | of the lo ter: | ocation |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is no instruction, it resets | ot repeatable. If this instruct the repeat counter (RPTC) | ion follows the and executes | e RPT s only o | nce. |
| Example | ; Push ; conte MOV ; MOV] | the contents of I ents of VarA: SP++,IER IER,@VarA | ER on the stack and loa ; Save IER on stack ; Load IER with cont | ad IER with ents of Var | the A | |

MOV loc16, #16bit

Save 16-bit Constant

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|---|--|---|-------------------|------|--|
| MOV loc16, #1 | 6bit | | 0010 1000 LLLL LLLL CCCC CCCC CCCC CCCC | Х | Y | N+1 | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | | |
| | #16bit | 16-bit constant imme | ediate value | | | | |
| Description | | Load the location pointed to by the "loc16" addressing mode with the 16-bit constant immediate value: | | | | | |
| | | [loc16] = 16bit; | | | | | |
| | | Note: For #16bit = #0, | see the MOV loc16, #0 instruction | n on page 6-166. | | | |
| | | Smart Encoding: If loc16 = AL or AH and #16bit is an 8-bit number, then the assembler will encode this instruction as MOVB AX, #8bit to improve efficiency. To override this, use the MOVW AX, #16bit alias instruction. | | | | | |
| Flags and Modes | Ν | If $(loc16 = @AX)$, then the load to AX is tested for a negative condition. The negative flag bit is set if bit 15 of AX is 1, otherwise it is cleared. | | | | | |
| | Z | If $(loc16 = @AX)$, then the load to AX is tested for a zero condition. The bit is set if the result of the operation on the AX register generates a 0 value, otherwise it is cleared. | | | | | |
| Repeat | | This instruction is no instruction, it resets | ot repeatable. If this instruct the repeat counter (RPTC) | tion follows the and executes | e RPT s only o | nce. | |
| Example | ; Initi ; int16 ; for(i ; Arra MOVL X RPT # MOV * | <pre>alize the contents Array1[N]; =0; i < N; i++) y1[i] = 0xFFFF; AR2,#Array1 :(N-1) XAR2++,#0xFFFF</pre> | s of Arrayl with 0xFFF ; XAR2 = pointe ; Repeat next i ; Array1[i] = 0 ; i++ | F: r to Array1 nstruction N xFFFF, | N time | S | |

MOV loc16, *(0:16bit)

Move Value

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|-----------------------|----------------------------------|--|--|--|--------------------------------|--------------------------------|
| MOV loc16, *(0:16bit) | | | 1111 0101 LLLL LLLL CCCC CCCC CCCC CCCC | Х | Y | N+2 |
| Operands | loc16 *(0:16bit) | Addressing mode Immediate direct i only (0x00000000 | (see Chapter 5) memory address, access lo to 0x0000FFFF) | w 64K range | of data | space |
| Description | | Move the content address "0:16bit" mode: [loc16] = [0x00 | of the location specified by into the location pointed to | the constant of by the "loc10 | direct m 6" addr | emory essing |
| Flags and Modes | Ν | If $(loc16 = @AX)$, then the load to AX is tested for a negative condition. The negative flag bit is set if bit 15 of AX is 1, otherwise it is cleared. | | | | on. The |
| | z | If (loc16 = @AX), t is set if the result of otherwise it is clea | then the load to AX is tested of the operation on the AX re ared. | for a zero cor egister genera | ndition. Ites a 0 | The bit value, |
| Repeat | | This instruction is then it will be ex data-memory add Only the lower 16 | repeatable. If the operation xecuted N+1 times. Wher lress is post-incremented b bits of the address are affe | n follows a RF n repeated, t y 1 during ea octed. | PT instr he "(0: ich rep | uction, 16bit)" etition. |
| | | <pre>; Copy the cont ; int16 Array1 ; int16 Array2 ; for(i=0; i < ; Array2[i] =</pre> | <pre>cents of Array1 to Arra [N]; // Located in 1 N]; N; i++) Array1[i];</pre> | y2: .ow 64K of c | lata sj | pace |
| Example | MOVL XAR RPT #(N MOV *XA | 22,#Array2 1-1) R2++,*(0:Array1 | ; XAR2 = pointer ; Repeat next in ; Array2[i] = An ; i++ | r to Array2 hstruction 1 rray1[i], | N time | ទ |

MOV loc16, #0

Clear 16-bit Location

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|---------------------------------------|---|--|---|-----------------------------|--------|-----|--|
| MOV loc16, #0 0010 1011 LLLL LLLL X Y | | | Υ | N+1 | | | |
| Operands | loc16 #0 | Addressing mode (s Immediate constant | ee Chapter 5) value of zero | | | | |
| Description | | Load the location pointed to by the "loc16" addressing mode with the value 0x0000: [loc16] = 0x0000; | | | | | |
| Flags and Modes | N Z | If (loc16 = @AX), the negative flag bit is so If (loc16 = @AX), the set if the result of th otherwise it is cleare | If (loc16 = $@AX$), then the load to AX is tested for a negative condition. The negative flag bit is set if bit 15 of AX is 1, otherwise it is cleared. If (loc16 = $@AX$), then the load to AX is tested for a zero condition. The bit is set if the result of the operation on the AX register generates a 0 value, otherwise it is cleared. | | | | |
| Repeat | | This instruction is repeatable. If the operation is follows a RPT instruction, then it will be executed N+1 times. | | | | | |
| Example | ; Initi ; int16 ; for(i ; Arra MOVL X RPT # MOV * | <pre>alize the content: Array1[N]; =0; i < N; i++) y1[i] = 0; AR2,#Array1 (N-1) XAR2++,#0</pre> | s of Arrayl with zero: ; XAR2 = pointer ; Repeat next ir ; Array1[i] = 0, ; i++ | r to Arrayl hstruction M | ∛ time | s | |

MOV loc16,ACC << 1..8

Save Low Word of Shifted Accumulator

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|----------------------|---------------------|--|--|-----------------|-----------------|----------|--|--|
| MOV loc16, ACC << 1 | | | 1011 0001 LLLL LLLL | 1 | Y | N+1 | | |
| MOV loc16, ACC << 28 | | | 0101 0110 0010 1101 0000 0SHF LLLL LLLL | 1 | Y | N+1 | | |
| | | | 1011 1SHF LLLL LLLL | 0 | - | 1 | | |
| Operands | loc16 ACC #18 | Addressing mode (s Accumulator registe Shift value | ee Chapter 5) r | | | | | |
| Description | | Load the content of with the low word of The ACC register is | Load the content of the location pointed to by the "loc16" addressing mode with the low word of the ACC register after left-shifting by the specified value. The ACC register is not modified: | | | | | |
| | | <pre>[loc16] = ACC >> <<18)</pre> | <pre>[loc16] = ACC >> (16 - shift value); [loc16] = low (ACC <<18)</pre> | | | | | |
| Flags and Modes | Ν | lf (loc16 = @AX), the The N flag is set if b | If (loc16 = $@AX$), then after the load AX is checked for a negative condition. The N flag is set if bit 15 of the AX is 1; else N is cleared. | | | | | |
| | Z | If (loc16 = @AX) the flag is set if AX is ze | n after the load AX is checke ro; else Z is cleared. | d for a zero co | ondition | . The Z | | |
| Repeat | | If the operation is repeatable, then the instruction will be executed $N+1$ times. The state of the Z and N flags will reflect the final result. If the operation is not repeatable, the instruction will execute only once. | | | | | | |
| Example | ; Mult ; VarC | Multiply two Q15 numbers (VarA and VarB) and store result in VarC as a Q15 number: | | | | | | |
| | MOV | T,@VarA | ; T = VarA | (Q15) | | | | |
| | MPY . | ACC,T,@VarB | ; ACC = VarA * VarB | (Q30) | | | | |
| | MOVH | @VarC,ACC << 1 | ; VarC = ACC >> (16- | -1) (Q15) | | | | |
| | ; VarC | as a Q31 number: | $- \Pi - Mar^{\Lambda}$ | (ന | - 014 | ` | | |
| | MPY | I, WVALA ACC, T, @VarB | ; I = VAIA ; ACC = VarA * VarB | (1 (A(| = Q14 CC = 0 | , 28) | | |
| | MOV | @VarC+0,ACC << 3 | ; VarC low = ACC << | < 3 | | _ • / | | |
| | MOVH | @VarC+1,ACC << 3 | ; VarC high = ACC >> | > (16-1) (Va | arC = | Q31) | | |
MOV loc16, ARn

Store 16-bit Auxiliary Register

| SY | NTAX | OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------------|--|--|--|---|-----------------------------------|----------------------------|
| MOV loc16, AF | ln | | 0111 11 | nnn LLLL LLLL | Х | 1 | 1 |
| Operands | loc16 ARn | Addressing mode | e (see Ch er 16 bits | apter 5) of auxiliary registers | | | |
| Description | | Load the contents [loc16] = ARn; If(loc16 = @ARn) is modified. The u | s of the 1 , then onl upper 16 | 6-bit location with AF y the lower 16 bits of the bits is unchanged. | n: he selected a | uxiliary r | egister |
| Flags and Modes | Ν | If (loc16 = @AX), of the AX register flag bit is set if the otherwise it is cle | then the is the sig operationared. | load to AX is tested for gn bit, 0 for positive, 1 on on the AX register o | r a negative c for negative generates a r | ondition. . The ne negative | Bit-15 gative value, |
| | Z | If (loc16 = @AX), set if the result o otherwise it is cle | f (loc16 = @AX), then the load to AX is tested for a zero condition. The bit is set if the result of the operation on the AX register generates a 0 value, otherwise it is cleared | | | | |
| Repeat | | This instruction is then it will be exe | s repeata ecuted N⊦ | ble. If the operation is 1 times. | s follows a R | PT instru | uction, |
| Example | MOV | @AL, AR3 | ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; | Load AL with the AR3. If bit 15 of N flag, else clea If AL is 0, set t | 16-bit con E AL is 1, ar it. che Z flag. | tents o set the | of ∍ |
| | MOV | @AR4,AR3 | ; ; ; | Load AR4 with the Upper 16 bits of unchanged. | e value in XAR4 are | AR3. | |
| | MOV | *SP++,AR3 | ; ; | Push the contents stack. Post incre | s of AR3 on ement SP. | to the | |
| | MOV | *XAR4++,AR4 | ; ; ; | Store contents of specified by XAR4 the contents of X | AR4 into 4. Post-inc KAR4. | locatio | on |
| | MOV | *XAR5,AR5 | ; ; ; | Pre-decrement the Store the content location specifie | e contents is of AR5 i ed by XAR5. | of XARS nto the | 5. 9 |

MOV loc16, AX

Store AX

| S | ΥΝΤΑΧ Ο | PTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|--|---|--|-----------------------|---------------------|
| MOV loc16, AX | | | 1001 | 011A LLLL LLLL | Х | Υ | N+1 |
| Operands | loc16 AX | Addressing mode (se | ee Cha H) or a | pter 5) ccumulator low (AL) | register | | |
| | | / loodinalator high (/ li | 1) OI U | | regiotor | | |
| Description | | Load the addressed I the 16-bit content of f [loc16] = AX; | ocatior the spe | n pointed to by the "lo ecified AX register (A | c16" addressi H or AL): | ng moo | de with |
| Flags and Modes | Ν | If (loc16 = @AX), the negative flag bit is se | f (loc16 = @AX), then the load to AX is tested for a negative condition. The negative flag bit is set if bit 15 of AX is 1, otherwise it is cleared. | | | | |
| | Z | If (loc16 = @AX), the set if the result of th otherwise it is cleared | n the lo le opei d. | bad to AX is tested for ration on the AX reg | r a zero condii ister generati | tion. Th es a 0 | ie bit is value, |
| Repeat | | If this operation follow The state of the N an | vs a RF nd Z fla | PT instruction, then it uses will reflect the fina | will be execute al result. | ed N+1 | times. |
| Example | ; Init: MOV MOVL RPT MOV | ialize all Array1 AH,#0xFFFF XAR2,#Array1 #9 *XAR2++, AH | eleme: ; ; ; ; | nts with the valu Load AH with the Load XAR2 with ac Repeat next inst Store contents of | e 0xFFFF: value 0xFF ddress of A cuction 10 t AH into 10 | FF rrayl times. | n |
| | | | ; ; | pointed by XAR2 a XAR2. | and post-ind | cremen | t |

MOV loc16, AX, COND

Store AX Register Conditionally

| | SYNTAX C | PTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------------|---|--|---|---|--|---------------------------------------|
| MOV loc16, A | X, COND | | | 0101 0110 0010 101A 0000 COND LLLL LLLL | 1 | - | 1 |
| Operands | loc16 AX COND | Address Accumu Conditio | sing mode (llator high (<i>i</i> onal codes: | see Chapter 5) AH) or accumulator low (AL) regis | ster | | |
| | | COND | Svntax | Description | Fla | as Test | ed |
| | | 0000 | NEQ | Not Equal To | Z = 0 | 9 | |
| | | 0001 | EQ | Equal To | Z = 1 | | |
| | | 0010 | GT | Greater Then | Z = 0 AN | ID N = 0 |) |
| | | 0011 | GEQ | Greater Then Or Equal To | N = 0 | | |
| | | 0100 | LT | Less Then | N = 1 | | |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 OF | R N = 1 | |
| | | 0110 | HI | Higher | C = 1 AN | 1D Z = 0 |) |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | |
| | | 1001 | LOS | Lower Or Same | C = 0 OF | R Z = 1 | |
| | | 1010 | NOV | No Overflow | V = 0 | | |
| | | 1011 | OV | Overflow | V = 1 | | |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 | | |
| | | 1101 | TC | Test Bit Set | TC = 1 | | |
| | | 1110 | NBIO | BIO Input Equal To Zero | BIO = 0 | | |
| | | 1111 | UNC | Unconditional | - | | |
| Description | | If the sp "loc16" a ister (AF if (CON Note: | ecified con addressing H or AL): ID = true Addressing r performs a p the condition | dition being tested is true, then the mode will be loaded with the cont) [loc16] = AX; modes are not conditionally executed re or post modification, the modification is true or not. | e location point cents of the spe d. Hence, if an a n will occur, rega | ed to by cified A addressir rdless of | r the X reg- ng mode whether |
| Flags and Modes | Ν | If (CONI the mov | D = true AN ve and if bi | ND loc16 = @AX), AX is tested t t 15 of AX is 1, the negative fla | for a negative ag bit is set. | conditic | on after |
| | Z | If (CON conditio | D = true A n and the : | ND loc16 = @AX), after the m zero flag bit is set if AX = 0, ot | ove, AX is tes herwise, it is | ted for cleared | a zero I. |

V If the V flag is tested by the condition, then V is cleared.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Example; Swap the contents of VarA and VarB if VarB is higher then VarA:MOVAL,@VarA; AL = VarA, XAR2 points to VarBMOVAH,@VarB; AH = VarB, XAR2 points to VarACMPAH,@AL; Compare AH and ALMOV@VarA,AH,HI; Store AH in VarA if higherMOV@VarB,AL,HI; Store AL in VarB if higher

MOV loc16,IER

Store Interrupt-Enable Register

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-------------------------------------|---|---|-----------------------------------|----------------------|-------------------|
| MOV loc16,IER | | | 0010 0000 LLLL LLLL | Х | - | 1 |
| Operands | loc16 IER | Addressing mode (s Interrupt enable regi | ee Chapter 5) ster | | | |
| Description | | Save the content of t addressing mode: [loc16] = IER; | the IER register in the location | on pointed to | by the ' | 'loc16" |
| Flags and Modes | N Z | If (loc16 = @AX) and If (loc16 = @AX) and cleared. | d bit 15 of AX is 1, then N is d the value of AX is zero, th | set; otherwise nen Z is set; o | e N is cl otherwi | eared. se Z is |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | PPT nce. |
| Example | ; Push ; conte MOV * MOV I | the contents of I ents of VarA: SP++,IER ER,@VarA | ER on the stack and loa ; Save IER on stack ; Load IER with conte | ad IER with nts of VarA | the | |

MOV loc16,OVC

Store the Overflow Counter

| - | | | | | | | | |
|---|--------|---|--|---------------------------------|---------------------|----------|--|--|
| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
| MOV loc16,OV | С | | 0101 0110 0010 1001 | 1 | - | 1 | | |
| | | | 0000 0000 LLLL LLLL | | | | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | | | |
| | 000 | Overnow counter | | | | | | |
| Description | | Store the 6 bits of the location pointed to by of the addressed loc | ore the 6 bits of the overflow counter (OVC) into the upper 6 bits of the ation pointed to by the "loc16" addressing mode and zero the lower 10 bits the addressed location: | | | | | |
| | | [loc16(15:10)] = [loc16(9:0)] = (| OVC;); | | | | | |
| Flage and N If (loc16 – $@AY$) and bit 15 of AY is 1, then set N: otherwise clear N | | | | | | I | | |
| Modes | z | If (loc16 = @AX) an | (loc16 = @AX) and AX is zero, then set Z; otherwise clear Z. | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | RPT nce. | | |
| Example | ; Sav | ve and restore con | tents of ACC and OVC | bits: | | | | |
| • | MOV | *SP++,OVC ; | Save OVC on stack | | | | | |
| | MOV | *SP++,AL ; | Save AL on stack | | | | | |
| | MOV | *SP++,AH ; | Save AH on stack | | | | | |
| | • | | | | | | | |
| | • | | | | | | | |
| | • | | | | | | | |
| | MOV | AH,*SP : | Restore AH from stack | | | | | |
| | MOV | AL,*SP ; | Restore AL from stack | | | | | |
| | MOV | OVC,*SP ; | Restore OVC from stack | | | | | |

| MOV loc16,P | | | 0011 1111 LLLL LLLL | Х | Y | N+1 |
|--------------------|--------|--|---|---|-------------------------------|----------------------------------|
| Operands | loc16 | Addressing mode (| see Chapter 5) | | | |
| | Ρ | Product register | | | | |
| Description | | The contents of the product shift mode the 16-bit location p is not modified by t | e P register are shifted by th (PM), and the lower half of the ointed to by the "loc16" addre he operation: | ne amount s e shifted valu essing mode. | pecified e is sto The P | 1 in the red into register |
| | | [loc16] = P << F | PM; | | | |
| Flags and Modes | Ν | lf (loc16 = @AX) ar otherwise, N is clea | nd bit 15 of the AX register is ared. | 1, then the | N bit is | set; |
| | Z | If (loc16 = @AX) ar set; otherwise Z is | If $(loc16 = @AX)$ and the value of AX after the load is zero, then the Z bit is set; otherwise Z is cleared. | | | |
| | РМ | The value in the PM product register. If operation), then th negative (arithmetic | value in the PM bits sets the shift mode for the output operation from the uct register. If the product shift value is positive (logical left shation), then the low bits are zero filled. If the product shift value tive (arithmetic right shift operation), the upper bits are sign extended | | | |
| Repeat | | This instruction is re it will be executed N final result. | epeatable. If the operation followed and the state of the Z | ows a RPT ir , and N flags | nstructic will ref | on, then lect the |
| Example | ; Calc | culate Y32 = M16*X3 | 16 >> 6 · т – м | | | |
| | MPY I | P,T,@X16 | ; P = T * X | | | |

OPCODE

MOV loc16,P

SYNTAX OPTIONS

SPM -6

MOV @Y32+0,P

MOVH @Y32+1,P

Store Lower Half of Shifted P Register

CYC

RPT

OBJMODE

; Set product shift to >> 6

; Y32 = P >> 6

MOV loc16, T

Store the T Register

| | ΞΥΝΤΔΧ Ο | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|---|--|---------------------------------|---------------------|---------------|--|--|
| MOV loc16 T | | i nono | | Y | | 1 | | |
| | | | | Χ | - | 1 | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | | | |
| | т | Upper 16 bits of the | multiplicand register (XT) | | | | | |
| Description | | Store the 16-bit T rea | gister contents into the locati | on pointed to | by the | "loc16" | | |
| | | [loc16] = T; | | | | | | |
| Flags and Modes | N | lf (loc16 = @AX) ar otherwise, N is clear | (loc16 = @AX) and bit 15 of the AX register is 1, then the N bit is set; nerwise, N is cleared. | | | | | |
| | Z | If (loc16 = @AX) and set; otherwise Z is c | d the value of AX after the lo leared. | oad is zero, th | en the | Z bit is | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. | | |
| Example | ; Calcu ; Y = (; X2 = ; X1 = | late using 16-bit (X0*C0) >> 2) + (X X1 X0 | multiply: 1*C1 >> 2) + (X2*C2 >> | 2) | | | | |
| | SPM -2 | ; | Set product shift to | >> 2 | | | | |
| | MOV T, | @X2 ; | T = X2 | | | | | |
| | MPI P, MOVD T | 1,@CZ ; r@X1 · | $P = 1^{2}$ | ~ 2 | | | | |
| | MPY P. | T.@C1 : | P = T*C1 | ~ 2 | | | | |
| | MOV @X | 2,T ; | X2 = X1 | | | | | |
| | MOVA I | r,@X0 ; | T = X0, ACC = X1*C1 > | > 2 + X2*C2 | >> 2 | | | |
| | MPY P, | T,@C0 ; | $P = T \star C0$ | | | | | |
| | MOV @X | X1,T ; | X1 = X0 | | | | | |
| | ADDL A | ACC, P << PM ; | ACC = X0*C0 >> 2 + X1 | *Cl >> 2 + | X2*C2 | >> 2 | | |
| | MOVL @ | PY,ACC ; | Store result into Y | | | | | |

MOV OVC, loc16

Load the Overflow Counter

| S | SYNTA) | K OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--|--|--|---------------------------------|------------------|---------------|--|--|--|
| MOV OVC, loc | 16 | | 0101 0110 0000 0010 | 1 | - | 1 | | | |
| | | | 0000 0000 LLLL LLLL | | | | | | |
| Operands | ovc | 6-bit overflow counter | ər | | | | | | |
| Description | | Load the overflow co to by the "loc16" add | Load the overflow counter (OVC) with the upper 6 bits of the location pointed to by the "loc16" addressing mode: | | | | | | |
| | | OVC = [loc16(15:1 | = [loc16(15:10)]; | | | | | | |
| Flags and Modes | ovc | The 6-bit overflow co | ounter is modified. | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. | | | |
| Example | ; Sat MOV MOV MOV MOV MOV | <pre>ve and restore contex *SP++,OVC ; *SP++,AL ; *SP++,AH ; AH,*SP ; AL,*SP ; OVC,*SP ;</pre> | nts of ACC and OVC bits Save OVC on stack Save AL on stack Save AH on stack Restore AH from stack Restore AL from stack Restore OVC from stack | 5: | | | | | |

MOV PH, loc16

Load the High Half of the P Register

| S | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------------------------|---|---|-----------------------------------|-------------------|------------------|
| MOV PH, loc16 | ; | | 0010 1111 LLLL LLLL | Х | - | 1 |
| Operands | PH loc16 | Upper 16 bits of the Addressing mode (s | product register (P) ee Chapter 5) | | | |
| Description | | Load the high 16 bits by the "loc16" addre | s of the P register (PH) with t ssing mode; leave the lowe | he 16-bit locat r 16 bits (PL) | tion poi uncha | nted to nged: |
| | | PH = [loc16]; PL = unchanged; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |
| Example | ; Swa MOV MOV MOV | p the contents of A PH,@AL ; L PL,@AH ; L ACC,@P ; L | H and AL: oad PH with AL oad PL with AH oad ACC with P (AH and | AL swapped |) | |

MOV PL, loc16

Load the Low Half of the P Register

| s | ΥΝΤΑΧ Ο | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-----------------------------------|---|---|------------------------------------|---------------------|-------------------|
| MOVL PL, loc1 | 6 | | 0010 0111 LLLL LLLL | Х | - | 1 |
| Operands | PL loc16 | Lower 16 bits of the Addressing mode (s | product register (P) ee Chapter 5) | | | |
| Description | | Load the high 16 bits by the "loc16" addre PL = [loc16]; PH = unchanged; | of the P register (PL) with ssing mode; leave the low | the 16-bit loca er 16 bits (PH) | tion poi i uncha | nted to inged: |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ir the repeat counter (RPTC) | struction follo | ows the s only o | e RPT ince. |
| Example | ; Swap MOV P MOV P MOV A | the contents of Al H,@AL ; Lo L,@AH ; Lo CC,@P ; Lo | H and AL: oad PH with AL oad PL with AH oad ACC with P (AH and | AL swapped |) | |

MOV PM, AX

Load Product Shift Mode

| 5 | Ο ΧΑΤΝΥ | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|--|--|---------------------|---------------|
| MOV PM, AX | | | 0101 0110 0011 100A | 1 | - | 1 |
| Operands | АХ | Accumulator high (/ | AH) or accumulator low (AL) | registers. | | |
| Description | | Load the product s register AX. PM = AX(2:0); | hift mode (PM) bits with the | e 3 least sigr | nificant | bits of |
| Flags and Modes | РМ | The product shift m AX. | ode bits are loaded with the | 3 least signif | icant bi | ts of |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this institute the repeat counter (RPTC) | struction follo and executes | ows the s only o | → RPT nce. |
| Example | ; Calcu CLRC A MOV AI ADDB A MOV PM MOV T, MPY P, MOVL A ADDL A MOVL @ | <pre>alate: Y32 = (M16* MODE ; AMODE ; AL,@Shift ; AL,#1 ; A,AX ; @X16 ; XT,@M16 ; ACC,@B32 ; ACC,P << PM ; @Y32,ACC ;;</pre> | <pre>X16 >> Shift) + B32, Shift) + B32, Shift) + B32, Shift = 0 Load AL with contents = 0 Load AL with contents = Convert "Shift" to PM Load PM bits with enco T = X16 P = X16*M16 ACC = B32 ACC = ACC + (P >> Shift Store result into Y32</pre> | nift = 0 to of "Shift" encoding ded "Shift" t) | 6 value | 2 |

MOV T, loc16

Load the Upper Half of the XT Register

| S | ΥΝΤΑΧ Ο | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|--|--|--|-------------------|---------------|
| MOV T, loc16 | | | 0010 1101 LLLL LLLL | Х | - | 1 |
| Operands | T loc16 | Upper 16 bits of the Addressing mode (s | multiplicand register (XT) ee Chapter 5) | | | |
| Description | | Load the T register v "loc16" addressing n T = [loc16]; | vith the 16-bit contents of th node: | e location poi | nted to | by the |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the only o | e RPT nce. |
| Example | ; Calcu; ; Y = (; X2 = ; X1 = SPM -: MOV T MPY P MOV @ MOVA (MPY P MOV @ MOVA (MPY P MOV @ ADDL (MOVL (| <pre>alate using 16-bit (X0*C0) >> 2) + (X) X1 X0 2 ; ,@X2 ; ,T,@C2 ; T,@X1 ; ,T,@C1 ; X2,T ; T,@X0 ; ,T,@C0 ; X1,T ; ACC,P << PM ; @Y,ACC ;</pre> | <pre>multiply: l*C1 >> 2) + (X2*C2 >> Set product shift to T = X2 P = T*C2 T = X1, ACC = X2*C2 >> P = T*C1 X2 = X1 T = X0, ACC = X1*C1 >> P = T*C0 X1 = X0 ACC = X0*C0 >> 2 + X1 Store result into Y</pre> | 2) >> 2 > 2 > 2 + X2*C2 *C1 >> 2 + | >> 2 X2*C2 | >> 2 |

MOV TL, #0

Clear the Lower Half of the XT Register

| S | SYNTAX OF | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---|--|---|---------------------------------|---------------------|---------------|--|--|
| MOV TL, #0 | | | 0101 0110 0101 0110 | 1 | - | 1 | | |
| Operands | т | Upper 16 bits of the | multiplicand register (XT) | | | | | |
| | #0 | Immediate constant | value of zero | | | | | |
| Description | | Load the lower half of the multiplicand register (TL) with zero, leaving the upper half (T) unchanged: | | | | | | |
| | | TL = 0x0000; T = unchanged; | | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. | | |
| Example | ; Calcui MOV TL MOV T, IMPYL P MOVL @ | late and keep low ,#0 ; @X16 ; ,XT,@M32 ; Y32,P ; | 32-bit result: Y32 = M TL = 0 T = X16 P = XT * M32 (high 32 Store result into Y32 | M32*X16 >> 3 | 32 ult) | | | |

MOV XARn, PC

Save the Current Program Counter

| SYNTAX OPTIONS | | | C | PCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--------|--|-------------------------------------|---|----------------|-------------------|---------------|--|--|--|
| MOV XARn, PC | | | 0011 111 | 0 0101 1nnn | 1 | - | 1 | | | |
| Operands | XARn | XAR0 to XAR7, 32 | -bit auxiliary | registers | | | | | | |
| | loc32 | Addressing mode (| dressing mode (see Chapter 5) | | | | | | | |
| | РС | 22-bit program cou | nter | | | | | | | |
| | | | | | | | | | | |
| Description | | Load XARn with the | e contents o | of the PC: | | | | | | |
| | | XARn = 0:PC; | | | | | | | | |
| | | | | | | | | | | |
| Flags and Modes | | None | | | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repea the repeat | table. If this inst counter (RPTC) a | ruction follow | ws the only or | e RPT nce. | | | |
| Example | TableA | : | ; Location of TableA is relative to | | | | | | | |
| | .long | CONST1 | ; | ; the current program | | | | | | |
| | .lo | ong CONST2 | | | | | | | | |
| | .lo | ong CONST3 | | | | | | | | |
| | • | | | | | | | | | |
| | FuncA: | YADE DO | | | | | | | | |
| | | XARS, PC XARS # (c Tablea) | | VARE - current | PC locatio | 'n | | | | |
| | MOMT. | ACC *+XAR5[2] | ; | XAR5 = TableA | start locat | ion | | | | |
| | MOVI | @VarA.ACC | : | Load ACC with | CONST2 | 1011 | | | | |
| | | | ; | Store CONST2 is | n VarA | | | | | |

MOVA T,loc16

Load T Register and Add Previous Product

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|------------|---|--|---|---|---|--|
| MOVA, T,loc16 | 3 | | 0001 0000 LLLL LLLL | Х | Y | N+1 | |
| Operands | T loc16 | Upper 16 bits of the Addressing mode (s | multiplicand register (XT) see Chapter 5) | | | | |
| Description | | Load the T register of "loc16" addressing r amount specified by of the ACC register: T = [loc16]; ACC = ACC + P << | with the 16-bit content of th node. Also, the content of th the product shift mode (PM) | e location poi ne P register, bits, is added | nted to shifted to the o | by the by the content | |
| Flags and Modes | N | After the operation, if N is cleared. | f bit 31 of the ACC register is | 1, the N bit is | set; oth | erwise, | |
| | Z | After the operation, i cleared. | f the value of ACC is zero, th | ne Z bit is set; | otherwi | ise Z is | |
| | С | If the addition gener | ates a carry, then C is set; | otherwise, C i | s clear | ed. | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected | | | |
| | OVC | If overflow mode is overflow, the counte operation generates | If overflow mode is disabled; and if the operation generates a positiv overflow, the counter is incremented. If overflow mode is disabled; and if th operation generates a negative overflow, the counter is decremented | | | | |
| | ΟνΜ | If overflow mode bit (0x7FFFFFFF) or overflows. | is set; the ACC value will maximum negative (0x80 | saturate maxi 000000) if t | mum p he op | ositive eration | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the product shift value is a low bits are zero filled. It right shift operation), the up | ne output oper positive (log the product oper bits are s | ration fr gical le shift va ign exte | om the ft shift alue is ended. | |
| Repeat | | This instruction is re be executed N+1 tin final result. The V fla | peatable. If the operation foll nes. The state of the Z, N, (ag will be set if an intermed | ows a RPT ins C and OVC fla iate overflow o | struction ags refl occurs. | n, it will ect the | |

| Example | ; Calcu ; Y = ; X2 = ; X1 = | ulate using 16-bi (X0*CO) >> 2) + (X1 X0 | t X1 | multiply: *C1 >> 2) + (X2*C2 >> 2) |
|---------|--------------------------------------|--|---------|--|
| | SPM | -2 | ; | Set product shift to >> 2 |
| | MOV | T,@X2 | ; | T = X2 |
| | MPY | P,T,@C2 | ; | $P = T^*C2$ |
| | MOVP | Τ,@X1 | ; | T = X1, ACC = $X2*C2 >> 2$ |
| | MPY | P,T,@C1 | ; | P = T*C1 |
| | MOV | @X2,T | ; | X2 = X1 |
| | MOVA | Τ,@ΧΟ | ; | T = X0, ACC = X1*C1 >> 2 + X2*C2 >> 2 |
| | MPY | P,T,@C0 | ; | P = T*C0 |
| | MOV | @X1,T | ; | X1 = X0 |
| | ADDL | ACC,P << PM | ; | ACC = X0*C0 >> 2 + X1*C1 >> 2 + X2*C2 >> 2 |
| | MOVL | @Y,ACC | ; | Store result into Y |

Load T Register

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|-------|--|--|---|--|---|--|--|--|
| MOVAD T, loc1 | 6 | | 1010 0111 LLLL LLLL | 1 | Ν | 1 | | | |
| Operands | т | Upper 16 bits of the | multiplicand register (XT) | | | | | | |
| | loc16 | Addressing mode (s | Idressing mode (see Chapter 5) | | | | | | |
| | | Note: For this operation @AH, @AL, @F | n, register-addressing modes canno PH, @PL, @SP, @T. An illegal inst | ot be used. The m ruction trap will b | odes are e genera | : @ARn, ated. | | | |
| Description | | Load the T register with the 16-bit content of the location pointed to by the "loc16" addressing mode and then load the next highest 16-bit location pointed to by "loc16" with the content of T. In addition, add the content of the P register, shifted by the amount specified by the product shift mode (PM) bits, to the content of the ACC register: | | | | | | | |
| | | T = [loc16]; [loc16 + 1] = T; ACC = ACC + P << | PM; | | | | | | |
| Flags and Modes | N | After the operation, otherwise, N is clear | if bit 31 of the ACC registe ed. | r is 1, then th | e N bit | is set; | | | |
| | Z | After the operation, if Z is cleared. | f the value of ACC is zero, th | en the Z bit is | set; oth | ierwise | | | |
| | С | If the addition gener | ates a carry, the C bit is set | ; otherwise, C | c is clea | ared. | | | |
| | v | If an overflow occurs | s, V is set; otherwise V is no | ot affected | | | | | |
| | ovc | If overflow mode is overflow, then the co the operation gene decremented. | disabled; and if the opera unter is incremented. If over erates a negative overflo | ation generat flow mode is c ow, then the | es a p lisablec e cour | ositive l; and if ıter is | | | |
| | OVM | If overflow mode bi positive (0x7FFFFF overflows. | t is set; then the ACC val F) or maximum negative (0 | ue will satura x80000000) if | ate ma the op | ximum eration | | | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the the product shift value is low bits are zero filled. If right shift operation), the up | ne output oper positive (log the product oper bits are s | ation fr gical le shift va ign exte | om the ft shift alue is ended. | | | |

| Repeat | | This instruction in instruction in instruction, it reserved. | s ts f | not r the re | epeatable. If this instruction follows the RPT epeat counter (RPTC) and executes only once. |
|---------|--|--|-------------|---------------------|---|
| Example | ; Calc ; Y = ; X2 = ; X1 = SPM | ulate using 16-bi (X0*CO) >> 2) + X1 X0 -2 | tt (X1 | mult 1*C1 Set | <pre>siply: >> 2) + (X2*C2 >> 2) product shift to >> 2</pre> |
| | MOVF MPYS MOVAD MPY | P,T,@C2 T,@X1 | , ; ; | P = T = | T*C2, ACC = 0 X1, ACC = X2*C2>>2, X2 = X1 T*C1 |
| | MOVAD MPY ADDL | T,@X0 P,T,@C0 ACC,P << PM | ; ; ; | T = P = ACC | X0, ACC = X1*C1>>2 + X2*C2>>2, X1 = X0 T*C0 = X0*C0>>2 + X1*C1>>2 + X2*C2>>2 |
| | MOVL | @Y,ACC | ; | Sto | re result into Y |

MOVB ACC,#8bit

Load Accumulator With 8-bit Value

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|--|---|---|--------------|-----|--|--|
| MOVB ACC,#8 | 3bit | | 0000 0010 CCCC CCCC | 1 | - | 1 | | |
| Operands | ACC #8bit | Accumulator regist 8-bit immediate un | Accumulator register 8-bit immediate unsigned constant value | | | | | |
| Description | | Load the ACC register with the specified 8-bit, zero-extended immediate constant: ACC = 0:8bit; | | | | | | |
| Flags and Modes | N Z | After the load, the After the load, the | After the load, the N flag is set if bit 31 of the ACC is 1, else N is cleared. After the load, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | | |
| Repeat | epeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Incre MOVB ; ADDL ; MOVL (| ement contents of ACC,#1 ACC,@VarA @VarA,ACC | 32-bit location VarA: ; Load ACC with the ; Add to ACC the cont ; Store result back : | value 0x000 tents of Va into VarA | 0 000: rA | 1 | | |

MOVB AR6/7, #8bit

Load Auxiliary Register With an 8-bit Constant

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-----------------|---------------------|---------|-----|-----|
| MOVB AR6, #8bit | 1101 0110 CCCC CCCC | Х | - | 1 |
| MOVB AR7, #8bit | 1101 0111 CCCC CCCC | Х | - | 1 |

| Operands | XARn | XAR6 OR XAR7, 32-bit auxiliary registers |
|--------------------|-------|---|
| | #8bit | 8-bit immediate constant value |
| Description | | Load AR6 or AR7 with an 8-bit unsigned constant and upper 16 bits of XAR6 and XAR7 are unchanged: |
| | | <pre>AR6/7 = 0:8bit; AR6/7H = unchanged;</pre> |
| Flags and Modes | | None |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once |

MOVB AX, #8bit

Load AX With 8-bit Constant

| S | YNTA) | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|-------------|---|---|---------------------------|---------------|-----|--|--|
| MOVB AX, #8 | bit | | 1001 101A CCCC CCCC | Х | _ | 1 | | |
| Operands | AX #8bit | Accumulator high (A 8-bit immediate cons | Accumulator high (AH) or accumulator low (AL) register | | | | | |
| Description | | Load accumulator hi unsigned 8-bit con accumulator register | Load accumulator high register (AH) or accumulator low register (AL) with an unsigned 8-bit constant zero extended, leaving the other half of the accumulator register unchanged: | | | | | |
| | | AX = 0:8bit; | | | | | | |
| Flags and Modes | N | Flag always set to z | ero. | | | | | |
| | Z | The load to AX is te results in AX = 0, oth | The load to AX is tested for a zero condition. The bit is set if the operation results in $AX = 0$, otherwise it is cleared. | | | | | |
| Repeat | | This instruction is instruction, it resets | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | MOVB CMP | AL, #0xF0 AL,*+XAR0[0] | ; Load AL with the ; Compare contents ; with AL. | value 0x00E pointed to | 70. by XAI | RO | | |
| | SB | Dest,EQ | ; Branch if values | are equal. | | | | |

MOVB AX.LSB, loc16

Load Byte Value

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | | |
|--------------------|------------|--|--|-------------------|----------|----------|--|--|--|--|
| MOVB AX.LSE | 3, loc16 | | 1100 011A LLLL LLLL | Х | - | 1 | | | | |
| Operands | AX.LS B | Least significant by (AL.LSB) register | east significant byte of accumulator high (AH.LSB) or accumulator low L.LSB) register | | | | | | | |
| | loc16 | Addressing mode (s | dressing mode (see Chapter 5) | | | | | | | |
| Description | | Load the least sign AL.LSB) with 8 bits mode. The most sig operand determines | _oad the least significant byte of the specified AX register (AH.LSB or AL.LSB) with 8 bits from the location pointed to by the "loc16" addressing mode. The most significant byte of AX is cleared. The form of the "loc16" operand determines which of its 8 bits are used to load AX.LSB: | | | | | | | |
| | | <pre>if(loc16 = *+XARA { if(offset is AX.LSB = [if(offset is AX.LSB = [} else AX.LSB = [loc AX.MSB = 0x00;</pre> | n[offset]) an even number) [loc16.LSB]; an odd value) [loc16.MSB]; 16.LSB]; | | | | | | | |
| | | Note: offset = 3-bit imr | nediate or AR0 or AR1 indexed a | ddressing modes | only. | | | | | |
| | | For the following ad | dress modes, the returned | result is unde | fined: | | | | | |
| | | *AR6%++ | (AMODE | = 0) | | | | | | |
| | | *0++ | (AMODE | = x) | | | | | | |
| | | *0 | (AMODE | = x) | | | | | | |
| | | *BR0++ | (AMODE | = x) | | | | | | |
| | | *BR0 | (AMODE | = x) | | | | | | |
| | | *0++, ARPn | (AMODE | = 1) | | | | | | |
| | | *0, ARPn | (AMODE | = 1) | | | | | | |
| | | *BR0++, AR | Pn (AMODE | = 1) | | | | | | |
| | | *BR0, AF | RPn (AMODE | = 1) | | | | | | |
| Flags and Modes | Z | After the move, AX AX = 0; otherwise it | is tested for a zero condition is cleared | on. The zero f | ag bit i | s set if | | | | |
| | N | After the move, AX is AX is 1; otherwise it | s tested for a negative cond is cleared. | ition. The bit is | set if b | it 15 of | | | | |

| Repeat | | This instruction is r instruction, it resets t | repeatable. repeat coun | . If this instruction follows the RPT ter (RPTC) and executes only once. |
|---------|--|---|--|---|
| Example | ; Swap ; Befo ; Afte MOVL MOVB MOVB MOVB MOVB MOVL | the byte order in ore operation: Var3 XAR2,#Var32 AL.LSB,*+XAR2[3] AH.LSB,*+XAR2[1] AL.MSB,*+XAR2[2] AH.MSB,*+XAR2[0] @Var32,ACC | <pre>e 32-bit "T = B3 B2 = B0 B1 ; Load XAF ; ACC(B0) ; ACC(B2) ; ACC(B1) ; ACC(B3) ; Store sv</pre> | <pre>Var32" location. B1 B0 B2 B3 R2 with address of "Var32" = Var32(B3), ACC(B1) = 0 = Var32(B1), ACC(B3) = 0 = Var32(B2), ACC(B1) = unch = Var32(B0), ACC(B1) = unch wapped result in "Var32"</pre> |

MOVB AX.MSB, loc16

Load Byte Value

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|------------|--|--|---|--|-----------------------------------|
| MOVB AX.MS | B, loc16 | | 0011 100A LLLL LLLL | Х | - | 1 |
| Operands | AX.MS B | Most significant byte (AL.MSB) register | e of accumulator high (Ał | H.MSB) or acc | cumulat | tor low |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Load the most sign AH.LSB) with 8 bits mode. The least sig "loc16" operand dete | ificant byte of the specifie from the location pointed to gnificant byte of AX is left formines which of its 8 bits | ed AX register to by the "loc1 unchanged. Th are used to loa | (AH.N 6" addr ne form ad AX.N | ISB or essing of the NSB |
| | | <pre>if(loc16 = *+XARr { if(offset is a AX.MSB = [if(offset is a AX.MSB = [} else AX.MSB = [loc: AX.LSB = unchange Note: offect 2 bitime</pre> | n[offset]) an even value) loc16.LSB]; an odd value) loc16.MSB]; ed; ed; | ddraasing modeo | only | |
| | | For the following add | dress modes, the returned | result is unde | fined: | |
| | | *AB6%++ | | = 0) | | |
| | | *0++ | (AMODE | = = v) | | |
| | | *0 | (AMODE | E = x) | | |
| | | *BR0++ | (AMODE | E = x) | | |
| | | *BR0 | (AMODE | E = x) | | |
| | | *0++, ARPn | (AMODE | E = 1) | | |
| | | *0, ARPn | (AMODE | = 1) | | |
| | | *BR0++, AR | Pn (AMODE | E = 1) | | |
| | | *BR0, AR | Pn (AMODE | E = 1) | | |
| Flags and Modes | N | After the move AX is set if bit 15 of AX is | tested for a negative conc 1; otherwise it is cleared. | lition. The neg | ative fla | ıg bit is |
| | Z | After the move, AX i AX = 0; otherwise it | s tested for a zero conditions is cleared. | on. The zero f | ag bit i | s set if |

| Repeat | | This instruction is instruction, it resets | not repeatable. If this instruction follows the RF the repeat counter (RPTC) and executes only once. | ъТ |
|---------|--|---|---|----|
| Example | ; Swag ; Befc ; Afte MOVL MOVB MOVB MOVB MOVB MOVL | o the byte order in pre operation: Variation: Variatio: Var | <pre>1 the 32-bit "Var32" location. :32 = B3 B2 B1 B0 :32 = B0 B1 B2 B3 ; Load XAR2 with address of "Var32" ; ACC(B0) = Var32(B3), ACC(B1) = 0 ; ACC(B2) = Var32(B1), ACC(B3) = 0 ; ACC(B1) = Var32(B2), ACC(B1) = unch ; ACC(B3) = Var32(B0), ACC(B1) = unch ; Store swapped result in "Var32"</pre> | |

MOVB loc16,#8bit,COND

Conditionally Save 8-bit Constant

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-----------------------|--|---------|-----|-----|
| MOVB loc16,#8bit,COND | 0101 0110 1011 COND CCCC CCCC LLLL LLLL | 1 | - | 1 |

| Operands | loc16 | Addressing mode (see Chapter 5) | | | | | | |
|--------------------|-------|----------------------------------|---|---|---|--|--|--|
| | #8bit | 8-bit immediate constant value | | | | | | |
| | COND | Conditio | onal codes | S: | | | | |
| | | COND | Syntax | Description | Flags Tested | | | |
| | | 0000 | NEQ | Not Equal To | Z = 0 | | | |
| | | 0001 | EQ | Equal To | Z = 1 | | | |
| | | 0010 | GT | Greater Then | Z = 0 AND $N = 0$ | | | |
| | | 0011 | GEQ | Greater Then Or Equal To | N = 0 | | | |
| | | 0100 | LT | Less Then | N = 1 | | | |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 OR N = 1 | | | |
| | | 0110 | HI | Higher | C = 1 AND $Z = 0$ | | | |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 | | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | | |
| | | 1001 | LOS | Lower Or Same | C = 0 OR Z = 1 | | | |
| | | 1010 | NOV | No Overflow | V = 0 | | | |
| | | 1011 | OV | Overflow | V = 1 | | | |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 | | | |
| | | 1101 | TC | Test Bit Set | TC = 1 | | | |
| | | 1110 | NBIO | BIO Input Equal To Zero | BIO = 0 | | | |
| | | 1111 | UNC | Unconditional | - | | | |
| Description | | If the sp constar | pecified co nt is stored | ondition being tested is true, then the till in the location pointed to by the "loc | he 8-bit zero extended c16" addressing mode: | | | |
| | | if(CON | D = true |) [loc16] = 0:8bit; | | | | |
| | | Note: / | Addressing r performs a p s true or not | nodes are not conditionally executed; there re- or post-modification, it will execute regard | fore, if an addressing mode lless of whether the condition | | | |
| Flags and Modes | Ν | If (CON negative is cleare | D = true A e conditior ed. | AND loc16 = @AX), then after the n n. The negative flag bit is set if bit 15 | nove AX is tested for a of AX is 1, otherwise it | | | |
| | Z | lf (CON zero co | D = true A ndition. Th | ND loc16 = @AX), then after the m ne zero flag bit is set if AX = 0, othe | nove, AX is tested for a erwise it is cleared. | | | |
| | v | If the V | flag is tes | ted by the condition, then V is clea | red. | | | |

| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |
|---------|--|
| Example | ; Calculate: ; if(VarA > 20) |

| ; Var | A = 0; | | | | | | |
|-------|------------------------|---|-------|-------|----|---------|------|
| CMP | @VarA,#20 | ; | Set : | flags | on | (VarA – | 20) |
| MOVB | <pre>@VarA,#0,GT</pre> | ; | Zero | VarA | if | greater | then |

MOVB loc16, AX.LSB

Store LSB of AX Register

| | SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------------|--|--|---|-----------------------------|-------------------------------|
| MOVB loc16, | AX.LSB | | 0011 110A LLLL LLLL | Х | - | 1 |
| Operands | loc16 AX.LS B | Addressing mode (s Least significant by (AL.LSB) register | ee Chapter 5) te of accumulator high (Al | H.LSB) or acc | cumulat | or low |
| Description | | Load 8 bits of the loc least significant byte form of the "loc16" of which of its 8 bits an | ation pointed to by the "loc1 of the specified AX register operand determines which e left unchanged: | 6" addressing r (AH.LSB or of its 8 bits aı | mode v AL.LSE e loade | vith the 3). The ed and |
| | | <pre>if(loc16 = *+XARr</pre> | <pre>n[offset]) is an even value) c16.LSB] = AX.LSB; c16.MSB] = unchanged; is an odd value) c16.LSB] = unchanged; c16.MSB] = AX.LSB; c16.MSB] = ax.LSB; nediate or AB0 or AB1 indexed as</pre> | daressina modes | only. | |
| | | This is a read modifi | write operation | | only. | |
| | | For the following ad | dress modes the returned | rocult is undo | finod | |
| | | | | | inieu. | |
| | | *AR6%++ | (AMODE | = 0) | | |
| | | *0++ | (AMODE | = X) | | |
| | | *0 | (AMODE | = X) | | |
| | | "DRU++ *PD0 | (AMODE | = X) | | |
| | | | | = X) - 1) | | |
| | | *0 APPn | | = 1) | | |
| | | 880±± Δ8 | | - 1) - 1) | | |
| | | *BB0 AF | RPn (AMODE | - 1) - 1) | | |
| | | | | '' | | |
| Flags and Modes | N | If (loc16 = @AX), the The negative flag bit | en after the move AX is test t is set if bit 15 of AX is 1, c | ted for a nega therwise it is | tive cor cleared | ndition. |
| | Z | If (loc16 = @AX), the zero flag bit is set if | en after the move, AX is test AX = 0, otherwise it is clea | ted for a zero red. | conditic | n. The |

| Repeat | This instruction is not repeatable. If this instruction follows the RP instruction, it resets the repeat counter (RPTC) and executes only once. |
|---------|--|
| Example | <pre>; Store the 32-bit contents of the ACC into the ; 32-bit contents of "Var32" location in reverse byte order: ; Before operation: ACC = B3 B2 B1 B0 ; After operation: Var32 = B0 B1 B2 B3 MOVL XAR2, #Var32 ; Load XAR2 with address of "Var32" MOVB *+XAR2[0],AH.MSB ; Var32(B0) = ACC(B3) MOVB *+XAR2[1],AH.LSB ; Var32(B1) = ACC(B2) MOVB *+XAR2[2],AL.MSB ; Var32(B2) = ACC(B1) MOVB *+XAR2[3],AL.LSB ; Var32(B3) = ACC(B0)</pre> |

MOVB loc16, AX.MSB

Store MSB of AX Register

| | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|------------|---|--|--|------------------------------|-------------------------------|
| MOVB loc16, | AX.MSB | | 1100 100A LLLL LLLL | Х | - | 1 |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| | AX.MS B | Most significant byt (AL.MSB) register | e of accumulator high (A | H.MSB) or acc | cumulat | tor low |
| Description | | Load 8 bits of the loc most significant byte form of the "loc16" of which of its 8 bits ar | ation pointed to by the "loc of the specified AX registe operand determines which e left unchanged: | 6" addressing r (AH.MSB or of its 8 bits a | mode v AL.MSE re loade | vith the 3). The ed and |
| | | <pre>if(loc16 = *+XARn { if(offset is [loc16.LSE [loc16.MSE if(offset is [loc16.LSE [loc16.LSE [loc16.MSE } else [loc16.LSE [loc16.MSE Note: offset = 3-bit imm</pre> | <pre>n[offset]) an even number)] = AX.MSB;] = unchanged; an odd number)] = unchanged;] = AX.MSB;] = AX.MSB;] = unchanged; mediate or AR0 or AR1 indexed and and and and and and and and and an</pre> | uddressing modes | only. | |
| | | This is a read-modif | v-write operation. | J. J | | |
| | | For the following ad | dress modes, the returned | result is unde | fined: | |
| | | *AR6%++ | (AMODI | E = 0) | | |
| | | *0++ | (AMODI | E = x) | | |
| | | *0 | (AMODI | E = x) | | |
| | | *BR0++ | (AMODI | E = x) | | |
| | | *BR0 | (AMODI | E = x) | | |
| | | *0++, ARPn | (AMODI | E = 1) | | |
| | | *0, ARPn | (AMODI | E = 1) | | |
| | | *BR0++, AR | Pn (AMODI | Ξ = 1) | | |
| | | *BR0, AF | RPn (AMODI | E = 1) | | |
| Flags and Modes | N | If (loc16 = @AX), the The negative flag bit | en after the move AX is te t is set if bit 15 of AX is 1, | sted for a nega otherwise it is | tive cor cleared | ndition. |
| | Z | If (loc16 = @AX), the zero flag bit is set if | en after the move, AX is tea AX = 0, otherwise it is clea | sted for a zero ared. | conditic | on. The |

| Repeat | This instruction is not repeatable. If this instruction follows the RF instruction, it resets the repeat counter (RPTC) and executes only once. | ۲ |
|---------|--|---|
| Example | <pre>; Store the 32-bit contents of the ACC into the ; 32-bit contents of "Var32" location in reverse byte order: ; Before operation: ACC = B3 B2 B1 B0 ; After operation: Var32 = B0 B1 B2 B3 MOVL XAR2, #Var32 ; Load XAR2 with address of "Var32" MOVB *+XAR2[0],AH.MSB ; Var32(B0) = ACC(B3) MOVB *+XAR2[1],AH.LSB ; Var32(B1) = ACC(B2) MOVB *+XAR2[2],AL.MSB ; Var32(B2) = ACC(B1) MOVB *+XAR2[3],AL.LSB ; Var32(B3) = ACC(B0)</pre> | |

MOVB XARn, #8bit

Load Auxiliary Register With 8-bit Value

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-------------------|---------------------|---------|-----|-----|
| MOVB XAR05, #8bit | 1101 Onnn CCCC CCCC | Х | - | 1 |
| MOVB XAR6, #8bit | 1011 1110 CCCC CCCC | 1 | - | 1 |
| MOVB XAR7, #8bit | 1011 0110 CCCC CCCC | 1 | - | 1 |

| Operands | XARn | XAR0 to XAR7, 32-bit auxiliary registers | | | | | |
|--------------------|-------|--|--|--|--|--|--|
| | #8bit | 8-bit immediate constant value | | | | | |
| Description | | Load XARn with the 8-bit unsigned immediate value: | | | | | |
| | | XARn = 0:8bit; | | | | | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | MOVB | XAR0, #F2h ; Load XAR0 with 0x0000 00F2 | | | | | |

MOVDL XT, loc16

Store XT and Load New XT

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--|---|--|-------------|--------|------|--|--|--|
| MOVDL XT,loc16 | | | 1010 0110 LLLL LLLL | 1 | Y | N+1 | | | |
| Operands | хт | Multiplicand register | | | | | | | |
| | 10C32 | Addressing mode (see Chapter 5) Note: For this operation, register-addressing modes cannot be used. The modes are: @XARn, @ACC, @P, @XT. An illegal instruction trap will be generated. | | | | | | | |
| Description | | Load the XT register with the 32-bit content of the location pointed to by the "loc32" addressing mode and then load the next highest 32-bit location pointed to by "loc32" with the content of XT: | | | | | | | |
| | | XT = [loc32]; [loc32 + 2] = XT | ; | | | | | | |
| Flags and Modes | | None | | | | | | | |
| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | | |
| Example | ; Calcu ; Y = | <pre>culate using 32-bit multiply, retaining high result: (X0*C0) >> 2) + (X1*C1 >> 2) + (X2*C2 >> 2) ; X2 = X1 ; X1 = X0</pre> | | | | | | | |
| | SPM | -2 | ; Set product shift to | >> 2 | | | | | |
| | ZAPA | VIII OVO | Zero ACC, P, OVC | | | | | | |
| | OMPYL | AI, @AZ P. XT. @C2 | | | | | | | |
| | MOVDL | XT,@X1 | ; XT = X1, ACC = X2*C2> | >2, X2 = X1 | | | | | |
| | QMPYAL | P,XT,@C1 | ; P = XT*C1 | | | | | | |
| | MOVDL | XT,@X0 | ; XT = X0, ACC = X1*C1> | >2 + X2*C2> | >2, X1 | = X0 | | | |
| | QMPYAL TODI. | P, XT, @CU | ; P = XT*C0 • ACC - X0*C0>>> + V1*C | 1550 ± ¥0*C | 2~~2 | | | | |
| | MOVL | @Y,ACC | ; Store result into Y | 1//2 T A2"C | | | | | |

MOVH loc16,ACC << 1..8

Save High Word of Shifted Accumulator

| SYNTAX OPTIONS | | | | OPCO | DE | OBJMODE | RPT | CYC | |
|-----------------------|---------------------|---|--|------------------------|---------------------|-------------|-------|------|--|
| MOVH loc16, ACC << 1 | | | 1013 | L 0011 LI | LL LLLL | 1 | Υ | N+1 | |
| MOVH loc16, ACC << 28 | | | 0103 | L 0110 00 D 0SHF LI |)10 1111 LL LLLL | 1 | Y | N+1 | |
| | | | 1013 | L OSHF LI | LL LLLL | 0 | - | 1 | |
| Operands | loc16 ACC #18 | Addressing mode (s Accumulator register Shift value | Addressing mode (see Chapter 5) Accumulator register Shift value | | | | | | |
| Description | | Load the content of the location pointed to by the "loc16" addressing mode with the high word of the ACC register after left-shifting by the specified value. The ACC register is not modified: [loc16] = ACC >> (16 - shift value); | | | | | | | |
| Flags and Modes | N | If (loc16 = $@AX$), then after the load AX is checked for a negative condition. The N flag is set if bit 15 of the AX is 1; else N is cleared. | | | | | | | |
| | Z | If $(loc16 = @AX)$ then after the load AX is checked for a zero condition. The Z flag is set if AX is zero; else Z is cleared. | | | | | | | |
| Repeat | | If the operation is repeatable, then the instruction will be executed $N+1$ times. The state of the Z and N flags will reflect the final result. If the operation is not repeatable, the instruction will execute only once. | | | | | | | |
| Example | ; Mult ; VarC | iply two Q15 numbers (VarA and VarB) and store result in 2 as a Q15 number: | | | | | | | |
| | MOV | T,@VarA | ; | T = VarA | Į | (Q15) | | | |
| | MPY | ACC,T,@VarB | ; | ACC = Va | arA * VarB | (Q30) | | | |
| | MOVH | @VarC,ACC << 1 | ; | VarC = P | ACC >> (16- | -1) (Q15) | | | |
| | ; VarC | as a Q31 number: | | | | · | | | |
| | MOV | T,@VarA | ; | T = VarA | A torp | (T) | = Q14 |) | |
| | MOV | @VarC+0.ACC << 3 | ; | VarC low | v = ACC << | (A) < 3 | | 20) | |
| | MOVH | @VarC+1,ACC << 3 | ; | VarC hig | gh = ACC > 2 | > (16-1) (V | arC = | Q31) | |

MOVH loc16, P

Save High Word of the P Register

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|---|--|---------|-----|-----|--|--|
| MOVH loc16,P | | | 0101 0111 LLLL LLLL | Х | Y | N+1 | | |
| Operands | loc16 P | Addressing mode (s Product register | ee Chapter 5) | | | | | |
| Description | | The contents of the P register are shifted by the amount specified in the product shift mode (PM), and the upper half of the shifted value is stored into the 16-bit location pointed to by the "loc16" addressing mode. The P register is not modified by the operation: | | | | | | |
| | | [loc16] = (P << PM) >> 16; | | | | | | |
| Flags and Modes | Ν | lf (loc16 = @AX) an otherwise, N is clea | If (loc16 = @AX) and bit 15 of the AX register is 1, then the N bit is set; otherwise, N is cleared. | | | | | |
| | Z | If (loc16 = @AX) and set; otherwise Z is c | If (loc16 = $@AX$) and the value of AX after the load is zero, then the Z bit is set; otherwise Z is cleared. | | | | | |
| | РМ | The value in the PM bits sets the shift mode for the output operation from the product register. If the product shift value is positive (logical left shift operation), then the low bits are zero filled. If the product shift value is negative (arithmetic right shift operation), the upper bits are sign extended. | | | | | | |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed $N+1$ times. The state of the Z, and N flags will reflect the final result. | | | | | | |
| Example | ; Calc MOV MPY SPM MOV MOVH | rulate Y32 = M16*X1 T,@M16 ; T P,T,@X16 ; P -6 ; S @Y32+0,P ; Y @Y32+1,P | 6 >> 6 | 6 | | | | |
MOVL ACC, loc32

Load Accumulator With 32 Bits

| s | YNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------------------------------|--|--|--------------------------------------|-------------|--------|
| MOVL ACC, loc | :32 | | 0000 0110 LLLL LLLL | Х | - | 1 |
| Operands | ACC loc32 | Accumulator register Addressing mode (s | ee Chapter 5) | | | |
| Description | | Load the ACC regis "loc32" addressing n ACC = [loc32]; | ter with the content of the node. | location poin | ited to | by the |
| Flags and Modes | N | After the load, the N | flag is set if bit 31 of the A0 | CC is 1, else l | N is cle | ared. |
| | Z | After the load, the Z | flag is set if the ACC is zer | o, else Z is cl | eared. | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | |
| Example | Calcul MOVL ADDL MOVL | ate the 32-bit valu ACC,@VarA ACC,@VarB @VarC,ACC | ne: VarC = VarA + VarB; ; Load ACC with con ; Add to ACC the co ; Store result into | ntents of Va Notents of V VarC | arA VarB | |

MOVL ACC,P << PM

Load the Accumulator With Shifted P

| 5 | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|-------------------------|---|---|---|---|---|--|--|
| MOVL ACC,P | << PM | | 0001 0110 1010 1100 | Х | - | 1 | | |
| Note: This instr | uction is a | n alias for the "MOVP T,loc1 | 6" operation with "loc16 = @T" ad | dressing mode. | | | | |
| Operands | ACC | Accumulator registe | ccumulator register | | | | | |
| | Р | Product register | | | | | | |
| | << PM | Product shift mode | | | | | | |
| Description | | Load the ACC regist by the product shift | oad the ACC register with the content of the P register shifted as specified y the product shift mode (PM): | | | | | |
| | | ACC = P << PM; | | | | | | |
| Flags and Modes | Ν | After the load, the N | After the load, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | |
| | Z | After the load, the Z | flag is set if the ACC is zer | o, else Z is cl | eared. | | | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the product shift value is the product shift value is a low bits are zero filled. If right shift operation), the up | ne output oper positive (log the product oper bits are s | ration fr gical le shift va ign exte | om the ft shift alue is ended. | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only a | e RPT | | |
| Example | ; Calc ; Yisa | culate: Y = Y + (M* 32-bit value, M and X are | X >> 4) • 16-bit values | | | | | |
| | SPM | -4 | ; Set product shift | t to >> 4 | | | | |
| | MOV | Τ,@Μ | ; T = M | | | | | |
| | MPY | Ρ,Τ,@Χ | ; P = M * X | | | | | |
| | MOVL | ACC, P << PM | ; ACC = M*X >> 4 | | | | | |
| | ADDL | @Y,ACC | ; $Y = Y + ACC$ | | | | | |

MOVL loc32, ACC

Store 32-bit Accumulator

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---------------------------------------|--|--|--------------------------------------|-------------|--------|--|
| MOVL loc32, A | VCC | | 0001 1110 LLLL LLLL | Х | - | 1 | |
| Operands | ACC loc32 | Accumulator register Addressing mode (s | ee Chapter 5) | | | | |
| Description | | Store the contents of "loc32" addressing n [loc32] = ACC; | of the ACC register into the node: | location poir | nted to | by the | |
| Flags and Modes | N | If (loc32 = @ACC) th is 1, else N is cleare | (loc32 = @ACC) then after the load, the N flag is set if bit 31 of the ACC 31 , else N is cleared. | | | | |
| | Z | If (loc32 = @ACC) th Z is cleared. | f (loc $32 = @ACC$) then after the load, the Z flag is set if ACC is zero, else Z is cleared. | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | Calcula MOVL A ADDL A MOVL @ | ate the 32-bit valu ACC,@VarA ACC,@VarB AVarC,ACC | ue: VarC = VarA + VarB; ; Load ACC with cor ; Add to ACC the co ; Store result into | ntents of Va Notents of V VarC | arA VarB | | |

MOVL loc32,ACC,COND

Conditionally Store the Accumulator

| SYNTAX OPTIONS | | | | OPCODE | OBJMODE | RPT | CYC |
|----------------|----------|-----------------------------------|--------------------------|---|-----------------------------------|--------------------|------------------|
| MOVL loc32, | ACC,CONE |) | | 0101 0110 0100 1000 0000 COND LLLL LLLL | Х | - | 1 |
| Operands | loc32 | Addres | sing mode | e (see Chapter 5) | | | |
| | ACC | Accum | ulator regi | ster | | | |
| | COND | Conditi | onal code | S: | | | |
| | | COND | Syntax | Description | Flag | gs Test | ed |
| | | 0000 | NEQ | Not Equal To | Z = 0 | | |
| | | 0001 | EQ | Equal To | Z = 1 | | |
| | | 0010 | GT | Greater Then | Z = 0 AN | ID N = (|) |
| | | 0011 | GEQ | Greater Then Or Equal To | N = 0 | | |
| | | 0100 | LT | Less Then | N = 1 | | |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 OF | R N = 1 | |
| | | 0110 | HI | Higher | C = 1 AN | VD Z = (|) |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | |
| | | 1001 | LOS | Lower Or Same | C = 0 OI | R Z = 1 | |
| | | 1010 | NOV | No Overflow | V = 0 | | |
| | | 1011 | OV | Overflow | V = 1 | | |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 | | |
| | | 1101 | тс | Test Bit Set | TC = 1 | | |
| | | 1110 | NBIO | BIO Input Equal To Zero | BIO = 0 | | |
| | | 1111 | UNC | Unconditional | - | | |
| Description | | If the sp the "loc register | becified co 32" addre | ndition being tested is true, the ssing mode will be loaded wi | en the location th the content | pointe ts of th | d to by e ACC |

if(COND = true) [loc32] = ACC;

Note: Addressing modes are not conditionally executed. Hence, if an addressing mode performs a pre or post modification, the modification will occur regardless of whether the condition is true or not.

| Flags and Modes | Ν | If (COND = true AN ACC is 1, N is set; | If (COND = true AND loc32 = @ACC), then after the move if bit 31 of ACC is 1, N is set; otherwise N cleared. | | | | | |
|--------------------|--|--|--|---|--|--|--|--|
| | Z | If (COND = true AN the Z bit is set; othe | D loc32 erwise i | = $@ACC$), then after the move if (ACC = 0), then t is cleared. | | | | |
| | V | If the V flag is teste | the V flag is tested by the condition, then V is cleared. | | | | | |
| Repeat | | This instruction is instruction, it resets | not re the rep | epeatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. | | | | |
| Example | ; Sway MOVL MOVL CMPL MOVL MOVL | ACC,@VarB P,@VarA ACC,@P @VarA,ACC,HI @P,ACC,HI @VarA,P | 32-bit ; ; ; ; ; ; | VarA and VarB if VarB is higher: ACC = VarB P = VarA Set flags on (VarB - VarA) VarA = ACC if higher P = ACC if higher VarA = P | | | | |

MOVL loc32,P

Store the P Register

| S | YNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|---|---------------------------------|------------------|---------------|
| MOVL loc32,P | | | 1010 1001 LLLL LLLL | 1 | - | 1 |
| Operands | loc32 | Addressing mode (s | ee Chapter 5) | | | |
| | Ρ | Product register | | | | |
| Description | cription Store the P register contents into the location pointed to by the "loc32 addressing mode: | | | | | |
| | | [loc32] = P; | | | | |
| Flags and Modes | Ν | If (loc32 = @ACC) a otherwise, N is clear | and bit 31 of the ACC register red. | er is 1, then th | ie N bit | is set; |
| | Z | If (loc32 = @ACC) ar set; otherwise Z is c | nd the value of ACC after the leared. | load is zero, tł | nen the | Z bit is |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |
| Example | ; Add | 64-bit VarA, VarB | and VarC, and store res | sult in VarI |): | |
| • | MOVL | P,@VarA+0 ; L | oad P with low 32 bits | of VarA | | |
| | MOVL | ACC,@VarA+2 ; L | oad ACC with high 32 b | its of VarA | | |
| | ADDUL | P,@VarB+0 ; A | dd to P unsigned low 32 | 2 bits of Va | arB | - |
| | ADDCL | ACC,@VarB+2 ; A | dd to ACC with carry h | igh 32 bits | ot Va | rB |
| | | P,@varc+0; A | dd to ACC with carry h | igh 22 hita | of Va | rC |
| | MOVL | @VarD+0.P : S | tore low 32-bit result | into VarD | JI VA | I C |
| | MOVL | <pre>@VarD+2,ACC ; S</pre> | tore high 32-bit result | t into VarD | | |

MOVL loc32, XARn

Store 32-bit Auxiliary Register

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| MOVL loc32, XAR0 | 0011 1010 LLLL LLLL | 1 | - | 1 |
| MOVL loc32, XAR1 | 1011 0010 LLLL LLLL | 1 | - | 1 |
| MOVL loc32, XAR2 | 1010 1010 LLLL LLLL | 1 | - | 1 |
| MOVL loc32, XAR3 | 1010 0010 LLLL LLLL | 1 | - | 1 |
| MOVL loc32, XAR4 | 1010 1000 LLLL LLLL | 1 | - | 1 |
| MOVL loc32, XAR5 | 1010 0000 LLLL LLLL | 1 | - | 1 |
| MOVL loc32, XAR6 | 1100 0010 LLLL LLLL | Х | - | 1 |
| MOVL loc32, XAR7 | 1100 0011 LLLL LLLL | Х | - | 1 |

| Operands | loc32 | Addressing mode | e (see Chapter 5) |
|--------------------|--------|---|--|
| | XARn | XAR0 to XAR7, 3 | 32-bit auxiliary registers |
| Description | | Load the conten XARn: | ts of the 32-bit addressed location with the contents of |
| | | [loc32] = XARr | 1; |
| Flags and Modes | Ν | If (loc32 = @ACC Bit-31 of the ACC negative flag bit negative value, c | C), then the load to ACC is tested for a negative condition. C register is the sign bit, 0 for positive, 1 for negative. The is set if the operation on the ACC register generates a otherwise it is cleared. |
| | Z | If (loc32 = @ACC is set if the result otherwise it is cle | c), then the load to ACC is tested for a zero condition. The bit of the operation on the ACC register generates a 0 value, eared |
| Repeat | | This instruction instruction, it res | is not repeatable. If this instruction follows the RPT ets the repeat counter (RPTC) and executes only once. |
| Example | MOVL @ | ACC, XAR0 | ; Move the 32-bit contents of XAR0 into ACC. ; If bit 31 of the ACC is 1 set N. If ; ACC = 0, set Z. |
| | MOVL * | XAR1, XAR7 | ; Move the 32-bit contents of XAR7 into the ; location pointed to by XAR1. |
| | MOVL * | *XAR6++,XAR6 | ; Move the 32-bit contents of XAR6 into the ; location pointed to by XAR6. Post-increment ; the contents of XAR6. |
| | MOVL * | XAR5, XAR5 | ; Predecrement the contents of XAR5. Move the ; 32-bit contents of XAR5 into the location ; pointed to by ; XAR5. |

MOVL loc32,XT

Store the XT Register

| 5 | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|------------------|--|--|---------------------------------|---------------------|----------------|--|--|
| MOVL loc32,X | Т | | 1010 1011 LLLL LLLL | 1 | - | 1 | | |
| Operands | loc32 XT | Addressing mode (s | dressing mode (see Chapter 5) Itiplicand register | | | | | |
| Description | | Store the T register mode: [loc32] = XT; | into 32-bit location pointed t | o by the "loc3 | 2" addi | ressing | | |
| Flags and Modes | N | lf (loc32 = @ACC) a otherwise, N is clea | and bit 31 of the ACC register red. | er is 1, then th | ne N bit | is set; | | |
| | Z | If (loc32 = @ACC) a set; otherwise Z is c | nd the value of ACC after the cleared. | load is zero, tl | hen the | Z bit is | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this insthe repeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT once. | | |
| Example | ; Calcu ; Y = | ulate using 32-bit (X0*CO) >> 2) + (X ; X2 = X1 ; X1 = X0 | multiply, retaining h 1*C1 >> 2) + (X2*C2 >> | igh result: 2) | | | | |
| | SPM | -2 | ; Set product shift to | >> 2 | | | | |
| | ZAPA | ; | ; Zero ACC, P, OVC | | | | | |
| | MOVL | XT,@X2 ; | XT = X2 | | | | | |
| | QMPYL | P, XT, @C2 ; | P = XT C2 | | | | | |
| | MUVL OMDVAT. | AI, @AI ; | $ACC = X1 = X1, ACC = X2^C2$ | >> 2 | | | | |
| | MOVI | @X2.XT | $X^{2} = X^{1}$ | | | | | |
| | MOVL | XT,@X0 | ; XT = X0, ACC = X1*C1 | >> 2 + X2*C | 2 >> 2 | 2 | | |
| | QMPYAL | P,XT,@C0 | P = XT*CO | | | | | |
| | MOVL | @X1,XT | ; X1 = X0 | | | | | |
| | ADDL | ACC, P << PM | ; ACC = X0*C0 >> 2 + X1 | *C1 >> 2 + | X2*C2 | >> 2 | | |
| | MOVL | @Y,ACC | ; Store result into Y | | | | | |

MOVL P,ACC

Load P From the Accumulator

| S | YNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|--|---|---------------------|---------------|
| MOVL P,ACC | | | 1111 1111 0101 1010 | Х | - | 1 |
| Operands | P ACC | Product register Accumulator registe | r | | | |
| Description | | Load the P register v P = ACC; | with the content of the ACC | register: | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | Calcu MOVL ABS MOVL MOVL ABS ADDL MOVL | late the 32-bit valu ACC,@VarA ACC P,ACC ACC,@VarB ACC ACC,@P @VarC,ACC | ue: VarC = abs(VarA) + ; Load ACC with cor ; Take absolute val ; Temp save ACC in ; Load ACC with cor ; Take absolute val ; Add contents of B ; Store result into | abs(VarB) ntents of Va lue of VarA P register ntents of Va lue of VarB 2 to ACC o VarC | arA arB | |

| u, | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------|--|---|---------------------------------|---------------------|----------------|
| MOVL P,loc32 | | | 1010 0011 LLLL LLLL | 1 | - | 1 |
| | | | | | | |
| Operands | Р | Product register | | | | |
| | loc32 | Addressing mode (s | ee Chapter 5) | | | |
| | | U | . , | | | |
| Description | | Load the P registe addressing mode: | r with the 32-bit location | pointed to b | y the | "loc32" |
| | | P = [loc32]; | | | | |
| | | | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT once. |
| Example | ; Add | 64-bit VarA, VarB | and VarC, and store res | sult in Varl |): | |
| • | MOVL | P,@VarA+0 ; L | oad P with low 32 bits | of VarA | | |
| | MOVL | ACC,@VarA+2 ; L | oad ACC with high 32 b: | its of VarA | | |
| | ADDUL | P,@VarB+0 ; A | dd to P unsigned low 32 | 2 bits of Va | arB | |
| | ADDCL | ACC,@VarB+2 ; A | dd to ACC with carry h | igh 32 bits | of Va | rB |
| | ADDUL | P,@VarC+0 ; A | dd to P unsigned low 32 | 2 bits of V | arC | |
| | ADDCL | ACC,@VarC+2 ; A | dd to ACC with carry h | igh 32 bits | of Va | rC |
| | MOVL | @VarD+0,P ; S | tore low 32-bit result | into VarD | | |
| | MOVL | <pre>@VarD+2,ACC ; S</pre> | tore high 32-bit result | t into VarD | | |

MOVL P,loc32

Load the P Register

MOVL XARn, loc32

Load 32-bit Auxiliary Register

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| MOVL XAR0, loc32 | 1000 1110 LLLL LLLL | 1 | - | 1 |
| MOVL XAR1, loc32 | 1000 1011 LLLL LLLL | 1 | - | 1 |
| MOVL XAR2, loc32 | 1000 0110 LLLL LLLL | 1 | - | 1 |
| MOVL XAR3, loc32 | 1000 0010 LLLL LLLL | 1 | - | 1 |
| MOVL XAR4, loc32 | 1000 1010 LLLL LLLL | 1 | - | 1 |
| MOVL XAR5, loc32 | 1000 0011 LLLL LLLL | 1 | - | 1 |
| MOVL XAR6, loc32 | 1100 0100 LLLL LLLL | Х | - | 1 |
| MOVL XAR7, loc32 | 1100 0101 LLLL LLLL | Х | - | 1 |

| Operands | XARn | XAR0 to XAR7, | 32-bit auxiliary registers | | | | | |
|--------------------|--------------|--------------------------------------|--|--|--|--|--|--|
| | loc32 | Addressing mod | e (see Chapter 5) | | | | | |
| Description | | Load XARn with XARn = [loc32] | Load XARn with the contents of the 32-bit addressed location: XARn = [loc32]; | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction instruction, it res | is not repeatable. If this instruction follows the RPT sets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | MOVL MOVL | XAR0,@ACC XAR2,*XAR0++ | ; Move the 32-bit contents of ACC into ; XAR0 ; Move the 32-bit value pointed to by ; XAR0 into XAR2. Post increment XAR0 | | | | | |
| | MOVL | XAR3,*XAR3++ | ; by 2 ; Move the 32-bit value pointed to by ; XAR3 into XAR3. Address modification ; of XAR3 is ignored. | | | | | |
| | MOVL | XAR4,*XAR4 | ; Predecrement the contents of XAR4. ; Move the 32-bit value pointed to by ; XAR4 into XAR4. | | | | | |

MOVL XARn, #22bit

Load 32-bit Auxiliary Register With Constant Value

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-------------------|--|---------|-----|-----|
| MOVL XAR0, #22bit | 1000 1101 00CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 1 |
| MOVL XAR1, #22bit | 1000 1101 01CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 1 |
| MOVL XAR2, #22bit | 1000 1101 10CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 1 |
| MOVL XAR3, #22bit | 1000 1101 11CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 1 |
| MOVL XAR4, #22bit | 1000 1111 00CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 1 |
| MOVL XAR5, #22bit | 1000 1111 01CC CCCC CCCC CCCC CCCC CCCC | 1 | - | 1 |
| MOVL XAR6, #22bit | 0111 0110 10CC CCCC CCCC CCCC CCCC CCCC | Х | - | 1 |
| MOVL XAR7, #22bit | 0111 0110 11CC CCCC CCCC CCCC CCCC CCCC | Х | - | 1 |

Operands XARn XAR0 to XAR7, 32-bit auxiliary registers

#22bit 22-bit immediate constant value

Description Load XARn with a 22-bit unsigned constant:

XARn = 0:22bit;

Flags and None Modes

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Example MOVL XAR4, #VarA

; Initialize XAR4 pointer with the ; 22-bit address of VarA

Load the XT Register

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|---|--|--|-------------------|----------------|
| MOVL XT, loc32 | 2 | | 1000 0111 LLLL LLLL | 1 | - | 1 |
| Operands | T loc32 | Upper 16 bits of the Addressing mode (s | multiplicand register (XT) ee Chapter 5) | | | |
| Description | | Load the XT register "loc32" addressing n XT = [loc32]; | with the 32-bit content of th node: | e location poi | nted to | by the |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the only o | e RPT Ince. |
| Example | ; Calcu ; Y = SPM ZAPA MOVL QMPYL MOVL QMPYAL MOVL QMPYAL MOVL ADDL MOVL | <pre>alate using 32-bit (X0*C0) >> 2) + (X) ; X2 = X1 ; X1 = X0 -2 ; XT,@X2 ; P,XT,@C2 ; XT,@X1 ; P,XT,@C1 ; @X2,XT ; XT,@X0 ; P,XT,@C0 ; @X1,XT ; ACC,P << PM ; @Y,ACC ;</pre> | <pre>multiply, retaining hi l*C1 >> 2) + (X2*C2 >> Set product shift to Zero ACC, P, OVC XT = X2 P = XT*C2 XT = X1, ACC = X2*C2 P = XT*C1 X2 = X1 XT = X0, ACC = X1*C1 P = XT*C0 X1 = X0 ACC = X0*C0 >> 2 + X1 Store result into Y</pre> | igh result: 2) >> 2 >> 2 >> 2 + X2*C *C1 >> 2 + 1 | 2 >> 2 X2*C2 | 2 >> 2 |

MOVP T,loc16

Load the T Register and Store P in the Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| MOVP T,loc16 | 0001 0110 LLLL LLLL | Х | - | 1 |

| Operands | т | Upper 16 bits of the | ne | multiplicand register (XT) |
|--------------------|-----------------|---|------------------|---|
| | loc16 | Addressing mode | (se | ee Chapter 5) |
| Description | | Load the T registe "loc16" addressing amount specified b register: | erv gm oyt | vith the 16-bit content of the location pointed to by the node. Also, the content of the P register, shifted by the the product shift mode (PM) bits, is loaded into the ACC |
| | | T = [loc16]; ACC = P << PM; | | |
| Flags and Modes | N | After the operation otherwise, N is cle | n if ear | bit 31 of the ACC register is 1, then the N bit is set; ed. |
| | Z | After the operation Z is cleared. | ı, if | the value of ACC is zero, then the Z bit is set; otherwise |
| | РМ | The value in the P product register. operation), then t negative (arithmet | M I If he | bits sets the shift mode for the output operation from the the product shift value is positive (logical left shift low bits are zero filled. If the product shift value is right shift operation), the upper bits are sign extended. |
| Repeat | | This instruction i instruction, it rese | s ts t | not repeatable. If this instruction follows the RPT the repeat counter (RPTC) and executes only once. |
| Example | ; Calc ; Y = | ulate using 16-b (X0*CO) >> 2) + ; X2 = X1 ; X1 = X0 | it (X | multiply: 1*C1 >> 2) + (X2*C2 >> 2) |
| | SPM | -2 | ; | Set product shift to >> 2 |
| | MOV | T,@X2 | ; | T = X2 |
| | MPY | P,T,@C2 | ; | $P = T^*C2$ |
| | MOVP | T,@X1 | ; | T = X1, ACC = X2*C2 >> 2 |
| | MPY | P,T,@C1 | ; | P = T*C1 |
| | MOV | @X2,T | ; | X2 = X1 |
| | MOVA | T,@X0 | ; | T = X0, ACC = X1*C1 >> 2 + X2*C2 >> 2 |
| | MPY | P, T, @C0 | ; | P = T * C0 |
| | MOV | WAL, T | ; | $X \perp = X \cup$ |
| | ADDL | ACC, P << PM | ; | $ACC = XU^*CU >> 2 + XI^*CI >> 2 + X2^*C2 >> 2$ |
| | MOVL | @Y,ACC | ; | Store result into Y |

MOVS T, loc16

Load T and Subtract P From the Accumulator

| SYNTAX OPTIONS | | | OF | CODE | OBJMODE | RPT | CYC |
|--------------------|------------|--|---|--|---|---|---|
| MOVS, T,loc16 | i | | 0001 0001 | LLLL LLLL | Х | Y | N+1 |
| Operands | T loc16 | Upper 16 bits of the Addressing mode (s | multiplicano ee Chapter | d register (XT) 5) | | | |
| Description | | Load the T register v "loc16" addressing n amount specified by content of the ACC r | with the 16- node. Also, the product egister: | bit content of th the content of th shift mode (PM | e location poi he P register,) bits, is subtra | nted to shifted acted fr | by the by the om the |
| | | T = [loc16]; ACC = ACC - P << | PM; | | | | |
| Flags and Modes | N | After the operation, otherwise, N is clear | if bit 31 of t ed. | he ACC registe | r is 1, then th | e N bit | is set; |
| | Z | After the operation, if Z is cleared. | the value o | f ACC is zero, th | ien the Z bit is | set; oth | erwise |
| | С | If the subtraction ger | ierates a bo | rrow, the C bit is | cleared; other | wise, C |) is set. |
| | V | If an overflow occurs | s, V is set; c | otherwise V is no | ot affected | | |
| | ovc | If overflow mode is overflow, then the co the operation gene decremented. | disabled; a unter is incr erates a r | and if the oper emented. If over negative overfl | ation generat flow mode is c ow, then the | es a p lisablec e cour | ositive l; and if nter is |
| | OVM | If overflow mode bi positive (0x7FFFFF overflows. | t is set; the F) or maxir | en the ACC va num negative (0 | lue will satura x80000000) if | ate ma the op | ximum eration |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the the produce low bits a right shift o | e shift mode for t t shift value is re zero filled. I peration), the up | he output oper positive (log f the product oper bits are s | ration fr gical le shift va ign exte | om the ft shift alue is ended. |
| Repeat | | This instruction is rep it will be executed N reflect the final resu occurs. | beatable. If t I+1 times. T Ilt. The V f | he operation foll he state of the lag will be set i | lows a RPT in: Z, N, C and (if an intermed | structio DVC fla liate ov | n, then ıgs will /erflow |

| Example | ; Calcu ; Y = ; X2 = ; X1 = | late using 16-bi (X0*C0) >> 2) + (2 X1 X0 | t X1 | multiply: *C1 >> 2) + (X2*C2 >> 2) |
|---------|--------------------------------------|--|---------|---|
| | SPM | -2 | ; | Set product shift to >> 2 |
| | MOVP | Τ,@X2 | ; | T = X2 |
| | MPYS | P,T,@C2 | ; | $P = T \star C2, ACC = 0$ |
| | MOVS | Τ,@X1 | ; | T = X1, ACC = $-X2*C2 >> 2$ |
| | MPY | P,T,@C1 | ; | P = T*C1 |
| | MOV | @X2,T | ; | X2 = X1 |
| | MOVA | Τ,@ΧΟ | ; | T = X0, ACC = -X1*C1 >> 2 - X2*C2 >> 2 |
| | MPY | P,T,@C0 | ; | P = T * C 0 |
| | MOV | @X1,T | ; | X1 = X0 |
| | SUBL | ACC,P << PM | ; | ACC = -X0*C0 >> 2 - X1*C1 >> 2 - X2*C2 >> 2 |
| | MOVL | @Y,ACC | ; | Store result into Y |

MOVU ACC,loc16

Load Accumulator With Unsigned Word

| S | | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|---|---|------------------|--------------------|
| MOVU ACC, lo | c16 | | 0000 1110 LLLL LLLL | Х | _ | 1 |
| Operands | ACC loc16 | Accumulator regist Addressing mode | ter (see Chapter 5) | | | |
| Description | | Load the low half addressed location high half of the acc | of the accumulator (AL) with n pointed to by the "loc16" ad cumulator (AH) with 0s: | the 16-bit co dressing moc | ntents le and | of the fill the |
| | | AH = 0x0000; | | | | |
| Flags and Modes | N | Clear flag. | | | | |
| | Z | After the load, the | Z flag is set if the ACC value | is zero, else z | Z is clea | ared. |
| Repeat | | This instruction is instruction is | s not repeatable. If this ins s the repeat counter (RPTC) a | truction follow and executes | ws the only o | RPT nce. |
| Example | ; Add MOVU ADD ADDU ADD ADDCU ADD | three 32-bit unsi ACC,@VarAlow ACC,@VarAhigh << ACC,@VarBlow ACC,@VarBhigh << ACC,@VarClow ACC,@VarChigh << | <pre>gned variables by 16-bit ; AH = 0, AL = VarA 16 ; AH = VarAhigh, AI ; ACC = ACC + 0:VarA 16 ; ACC = ACC + VarBA ; ACC = ACC + VarCA 16 ; ACC = ACC + VarCA</pre> | parts: Alow L = VarAlow rBlow nigh << 16 low + Carry nigh << 16 | | |

MOVU loc16,OVC

Store the Unsigned Overflow Counter

| | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|--|---|---------------------------------|--------------------|------------------|
| MOVU loc16,0 | OVC | | 0101 0110 0010 1000 | 1 | - | 1 |
| | | | 0000 0000 LLLL LLLL | | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| | OVC | Overflow counter | | | | |
| Description | | Store the 6 bits of the location pointed to be bits of the addressed | ne overflow counter (OVC) by the "loc16" addressing m d location: | into the lowe ode and zero | r 6 bits the up | of the per 10 |
| | | [loc16(15:6)] = [loc16(5:0)] = | 0; OVC; | | | |
| Flags and Modes | N | If (loc16 = @AX) an | d bit 15 of AX is 1, then set | N; otherwise | clear N | I. |
| | Z | If (loc16 = @AX) an | d AX is zero, then set Z; oth | nerwise clear | Z. | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ws the only o | RPT nce. |
| Example | ; Save | and restore con | itents of ACC and OVC | bits: | | |
| - | MOVU | *SP++,OVC ; | Save OVC on stack | | | |
| | MOV | *SP++,AL ; | Save AL on stack | | | |
| | MOV | *SP++,AH ; | Save AH on stack | | | |
| | • | | | | | |
| | | | | | | |
| | • | | | | | |
| | • | | | | | |
| | MOV | AH,*SP ; | Restore AH from stack | | | |
| | MOV | AL,*SP ; | Restore AL from stack | | | |
| | MOVU | OVC,*SP ; | Restore OVC from stack | | | |

MOVU OVC,loc16

Load Overflow Counter With Unsigned Value

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|--|---------------------------------|--------------------|----------------|
| MOVU OVC,lo | c16 | | 0101 0110 0110 0010 0000 0000 LLLL LLLL | 1 | - | 1 |
| Operands | OVC | 6-bit overflow count | er | | | |
| Description | | Load the overflow co to by the "loc16" add | ounter (OVC) with the lower (dressing mode: | 6 bits of the lo | cation p | pointed |
| | | OVC = [10C16(5:0)] |)] | | | |
| Flags and Modes | ovc | The 6-bit overflow c | ounter is modified. | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the s only o | e RPT once. |
| Example | ; Save MOVU MOV MOV MOV MOVU | <pre>and restore contex *SP++,OVC ; *SP++,AL ; *SP++,AH ; AH,*SP ; AL,*SP ; OVC,*SP ;</pre> | nts of ACC and OVC bits Save OVC on stack Save AL on stack Save AH on stack Restore AH from stack Restore AL from stack Restore OVC from stack | 3. | | |

MOVW DP, #16bit

Load the Entire Data Page

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|-----------------|--------|--|--------------|-----|-----|--|
| MOVW DP, #16bit | | 0111 0110 0001 1111 CCCC CCCC CCCC CCCC | X | - | 1 | |
| Operands | DP | Data page register | | | | |
| | #16bit | 16-bit immediate co | nstant value | | | |

Description Load the data page register with a 16-bit constant:

DP(15:0) = 16bit;

None

Flags and Modes

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

| Example | MOVW | DP, | #VarA | ; | Load DP with the data page that |
|---------|------|-----|---------|---|---------------------------------------|
| | | | | ; | contains VarA. Assumes VarA is in the |
| | | | | ; | lower 0x003F FFC0 of memory |
| | MOVW | DP, | #0F012h | ; | Load DP with data page number 0xF012 |

MOVX TL, loc16

Load Lower Half of XT With Sign Extension

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|----------------------------------|---|---|---------------------------------|------------------|---------------|--|
| MOVX TL,loc1 | 6 | | 0101 0110 0010 0001 xxxx xxxx LLLL LLLL 1 - 1 | | | | |
| Operands | TL | Lower 16 bits of the | multiplicand register (XT) | | | | |
| | loc32 | Addressing mode (s | Addressing mode (see Chapter 5) | | | | |
| Description | | Load the lower 16 bits of the multiplicand register (TL) with the 16-bit contents of the location pointed to by the "loc16" addressing mode and then sign extend that value into the upper upper 16 bits of XT: | | | | | |
| | | TL = [loc16]; T = sign extensio | on of TL; | | | | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. | |
| Example | ; Calcu MOVX IMPYL MOVL | TL,@X16 ; P,XT,@M32 ; @Y32,P ; | 32-bit result: Y32 = M XT = S:X16 P = XT * M32 (low 32 Store result into Y32 | 132*X16 bits of re | sult) | | |

MOVZ ARn, loc16

Load Lower Half of XARn and Clear Upper Half

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|---------------------|---------|-----|-----|
| MOVZ AR05, loc16 | 0101 1nnn LLLL LLLL | Х | I | 1 |
| MOVZ AR6, loc16 | 1000 1000 LLLL LLLL | 1 | 1 | 1 |
| MOVZ AR7, loc16 | 1000 0000 LLLL LLLL | 1 | - | 1 |

| Operands | ARn | AR0 to AR7, lower 16 bi | ts | of auxiliary registers |
|--------------------|--------------------------------|--|-----------|--|
| | loc16 | Addressing modes (See | cł | napter 5) |
| Description | | Load ARn with the conte ARn = [loc16]; ARnH = 0; | ent | s of the 16-bit location and clear ARnH: |
| Flags and Modes | | None | | |
| Repeat | | This instruction is not instruction, it resets the | re rep | epeatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. |
| Example | MOVL XAJ MOVZ AR MOVZ AR | R7, #ArrayA 0, *+XAR2[0] 7, *-SP[1] | ;;;;; | <pre>Initialize XAR2 pointer Load 16-bit value pointed to by XAR2 into AR0. XAR0(31:16) = 0. Load the first 16-bit value off of the stack into AR7. XAR7(31:16) = 0.</pre> |

MOVZ DP, #10bit

Load Data Page and Clear High Bits

| S | YNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------------|--|---|---------------------------------|--------------------|---------------|
| MOVZ DP, #10 | bit | | 1011 10CC CCCC CCCC | 1 | - | 1 |
| Operands | DP #10bit | Data page register 10-bit immediate col | nstant value | | | |
| Description | | Load the data page r | register with a 10-bit constar | nt and clear the | e upper | 6 bits: |
| | | DP(9:0) = 10bit; DP(15:10) = 0; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the sonly o | e RPT nce. |
| Example | MOVZ | DP, #VarA ; ; ; | Load DP with the data y VarA. Assumes VarA is 0x0000 FFC0 of memory | page that co in the low | ontain er | S |
| | MOVZ | DP, #3FFh ; | Load DP with page numb | er 0x03FF. | | |

MPY ACC,loc16, #16bit

16 X 16-bit Multiply

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|------------------------------|--|--|-----------------|--------------------|--------------------|
| MPY ACC, loc | 16,#16bit | | 0011 0100 LLLL LLLL CCCC CCCC CCCC CCCC | Х | - | 1 |
| Operands | ACC | Accumulator regist | er | | | |
| | loc16 | Addressing mode (| (see Chapter 5) | | | |
| | #16bit | 16-bit immediate c | onstant value | | | |
| Description | | Load the T register with the 16-bit content of the location pointed to by the "loc16" addressing mode; then, multiply the signed 16-bit content of the register by the specified signed 16-bit constant value: | | | | by the of the T |
| | | T = [loc16]; ACC = signed T | * signed 16bit; | | | |
| Flags and Modes | z | After the operation | , the Z flag is set if the ACC | is zero, else | Z is cle | eared. |
| | Ν | After the operation, | the N flag is set if bit 31 of the | e ACC is 1, els | e N is c | leared. |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in s the repeat counter (RPTC | struction follo | ows the es only | e RPT once. |
| Example | ; Calcul ; Y32 = MPY 2 | late signed using Y32 + X16 * 2000 ACC,@X16,#2000 | 16-bit multiply: ; T = X16, ACC = X16 | * 2000 | | |

MPY ACC, T, loc16

16 X 16-bit Multiply

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|--|--|---------------------------------|--------------------|---------------|
| MPY ACC,T,Io | c16 | 0001 0010 LLLL LLLL X - | | | - | 1 |
| Operands | ACC | Accumulator registe | r | | | |
| | т | Multiplicand register | | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Multiply the signed contents of the locati the result in the ACC | ultiply the signed 16-bit content of the T register by the signed 16-bit ontents of the location pointed to by the "loc16" addressing mode and store e result in the ACC register: | | | |
| | | ACC = signed T $*$ | ACC = signed T * signed [loc16]; | | | |
| Flags and Modes | z | After the operation, t | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. | | | |
| | Ν | After the operation, t | the N flag is set if bit 31 of t | he ACC is 1, | else N | is |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the sonly o | e RPT nce. |
| Example | ; Calcu ; Y32 = MOV MPY ADDL | late signed using Y32 + X16*M16 T,@X16 ; ACC,T,@M16 ; @Y32,ACC ; | 16-bit multiply: T = X16 ACC = T * M16 Y32 = Y32 + ACC | | | |

MPY P,loc16,#16bit

16 X 16-Bit Multiply

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|---|---|---|---------------------|---------------|--|
| MPY P,loc16,# | 16bit | | 1000 1100 LLLL LLLL CCCC CCCC CCCC CCCC | 1 | - | 1 | |
| Operands | Р | Product register | | | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | | |
| | #16bit | 16-bit immediate co | 16-bit immediate constant value | | | | |
| Description | | Multiply the signed 16-bit contents of the location pointed to by the "loc16" addressing mode by the 16-bit immediate value and store the 32-bit result in the P register: | | | | | |
| | | P = signed [loc10 | 6] * signed 16bit; | | | | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. | |
| Example | ; ; Cal ; Y = 0 ; C0, C SPM MOVB MPY MPYA MPYA ADDL MOVL | culate using 16-b (X0*C0) >> 2) + (X C1 and C2 are cons -2 ; ACC, #0 ; P,@X2,#C2 ; P,@X1,#C1 ; P,@X0,#C0 ; ACC, P << PM ; | <pre>it multiply: 1*C1 >> 2) + (X2*C2 >> tants Set product shift to >: Zero ACC P = X2*C2 ACC = X2*C2>>2, P = X1 ACC = X1*C1>>2 + X2*C2: ACC = X0*C0>>2 + X1*C1: Store result into X</pre> | 2), > 2 *C1 >>2, P = X0 >>2 + X2*C2 | *C0 | | |

MPY P,T,loc16

16 X 16 Multiply

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|---|---|---------------------|-------------------|
| MPY P,T,loc16 | | | 0011 0011 LLLL LLLL | Х | - | 1 |
| Operands | P T loc16 | Product register Multiplicand registe Addressing mode (| er see Chapter 5) | | | |
| Description | | Multiply the signed contents of the loca the 32-bit result in | I 16-bit content of the T reg tion pointed to by the "loc16" the P register: | gister by the addressing m | signed Iode an | 16-bit d store |
| | | P = signed T * s | signed [loc16]; | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | ; Calcu ; Y = (X ; X SPM MOVP MPYS MOVAD MPY MOVAD MPY ADDL MOVL | <pre>allate using 16-bi X0*C0) >> 2) + (X 2 = X1 1 = X0 -2 ; T,@X2 ; P,T,@C2 ; T,@X1 ; P,T,@C1 ; P,T,@C0 ; ACC,P << PM ; @Y,ACC ;</pre> | <pre>t multiply: 1*C1 >> 2) + (X2*C2 >> T = X2 P = T*C2, ACC = 0 T = X1, ACC = X2*C2>>2 P = T*C1 T = X0, ACC = X1*C1>>2 P = T*C0 ACC = X0*C0>>2 + X1*C1 Store result into Y</pre> | 2) > 2 , X2 = X1 + X2*C2>>2 >>2 + X2*C2 | , X1 = >>2 | XO |

MPYA P,loc16,#16bit

16 X 16-Bit Multiply and Add Previous Product

| Ś | SYNTAX OPTIONS OPCODE | | | OBJMODE | RPT | CYC |
|--------------------|-----------------------|---|---|---|---|---|
| MPYA P,loc16 | ,#16bit | | 0001 0101 LLLL LLLL CCCC CCCC CCCC CCCC | × | - | 1 |
| Operands | Р | Product register | | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| | #16bit | 16-bit immediate co | 16-bit immediate constant value | | | |
| Description | | Add the previous product (stored in the P register), shifted as specified by the product shift mode (PM) bits, to the ACC register. Load the T register with the content of the location pointed to by the "loc16" addressing mode. Multiply the signed 16-bit content of the T register by the signed 16-bit constant value and store the 32-bit result in the P register: | | | | by the vith the Aultiply t value |
| | | ACC = ACC + P << T = [loc16]; P = signed T * s: | PM; igned 16bit; | | | |
| Flags and Modes | Z | After the operation, | the Z flag is set if the ACC | is zero, else Z | Z is clea | ared. |
| | Ν | After the addition, the | e N flag is set if bit 31 of the | ACC is 1, else | e N is c | leared. |
| | С | If the addition gener | ates a carry, C is set; other | wise C is clea | red. | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | |
| | OVC | If overflow mode is overflow, then the co the operation gen decremented. | disabled; and if the oper punter is incremented. If over erates a negative overfl | ation generat flow mode is c ow, then the | es a p lisablec e cour | ositive I; and if nter is |
| | ΟνΜ | If overflow mode bit positive (0x7FFFFF operation overflowe | is set; then the ACC value FF) or maximum negative (d. | will saturate r 0x80000000) | naximu if the | m |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the product shift value is a low bits are zero filled. It right shift operation), the up | he output oper positive (log f the product oper bits are s | ration fr gical le shift va ign exte | om the ft shift alue is ended. |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only a | e RPT ince. |

Example ; Calculate using 16-bit multiply: ; Y = (X0*C0) >> 2) + (X1*C1 >> 2) + (X2*C2 >> 2), ; C0, C1 and C2 are constants SPM -2 ; Set product shift to >> 2 ; Zero ACC MOVB ACC,#0 MPY P,@X2,#C2 ; P = X2*C2 P,@X1,#C1 ; ACC = X2*C2>>2, P = X1*C1 MPYA MPYA P,@X0,#C0 ; ACC = X1*C1>>2 + X2*C2>>2, P = X0*C0 ACC,P << PM ADDL ; ACC = X0*C0>>2 + X1*C1>>2 + X2*C2>>2 @Y,ACC MOVL ; Store result into Y

MPYA P,T,loc16

16 X 16-bit Multiply and Add Previous Product

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-------|---|--|---|--|---|
| MPYA P,T,loc1 | 6 | | 0001 0111 LLLL LLLL | Х | Y | N+1 |
| Operands | Р | Product register | | | | |
| | т | Multiplicand register | • | | | |
| | loc16 | Addressing mode (s | ddressing mode (see Chapter 5) | | | |
| Description | | Add the previous pro product shift mode content of T by the "loc16" addressing r | Add the previous product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. Multiply the signed 16-bit content of T by the signed 16-bit content of the location pointed to by the "loc16" addressing mode and store the 32-bit result in the P register: | | | |
| | | ACC = ACC + P << P = signed T * | ACC = ACC + P << PM; P = signed T * signed [loc16]; | | | |
| Flags and Modes | Z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. | | | | ared. |
| | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | |
| | С | If the addition gener | ates a carry, C is set; other | wise C is clea | red. | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | |
| | OVC | If overflow mode is overflow, then the co the operation gen decremented. | disabled; and if the oper punter is incremented. If over erates a negative overfl | ation generat flow mode is c ow, then the | es a p lisablec e cour | ositive l; and if nter is |
| | ΟνΜ | If overflow mode bi positive (0x7FFFFf overflowed. | it is set; then the ACC va FF) or maximum negative (0 | lue will satura x80000000) if | ate ma the op | ximum eration |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the the product shift value is a low bits are zero filled. It right shift operation), the up | he output oper positive (log f the product oper bits are s | ation fr gical le shift va ign exte | om the ft shift alue is ended. |
| Repeat | | This instruction is re it will be executed N reflect the final resu occurs. | peatable. If the operation foll I+1 times. The state of the ult. The V flag will be set i | lows a RPT in Z, N, C and (if an intermed | structio DVC fla liate ov | n, then ags will /erflow |

| ; Calcu ; Y = | ulate using 16-b (X0*C0) >> 2) + | it (X | : multiply: X1*C1 >> 2) + (X2*C2 >> 2) |
|------------------|--|--|---|
| SPM | -2 | ; | Set product shift to >> 2 |
| MOVP | T,@X2 | ; | ACC = P, T = X2 |
| MPYS | P,T,@C2 | ; | ACC = ACC - P = 0, P = T*C2 |
| MOV | T,@X1 | ; | T = X1 |
| MPYA | P,T,@C1 | ; | ACC = $X2*C2>>2$, P = T*C1 |
| MOV | Τ,@ΧΟ | ; | T = X0 |
| MPYA | P,T,@C0 | ; | ACC = X1*C1>>2 + X2*C2>>2, P = T*C0 |
| ADDL | ACC,P << PM | ; | ACC = X0*C0>>2 + X1*C1>>2 + X2*C2>>2 |
| MOVL | @Y,ACC | ; | Store result into Y |
| | ; Calcu ; Y = SPM MOVP MPYS MOV MPYA ADDL MOVL | ; Calculate using 16-b ; Y = (X0*C0) >> 2) + SPM -2 MOVP T,@X2 MPYS P,T,@C2 MOV T,@X1 MPYA P,T,@C1 MOV T,@X0 MPYA P,T,@C0 ADDL ACC,P << PM MOVL @Y,ACC | ; Calculate using 16-bit ; Y = (X0*C0) >> 2) + (X SPM -2 ; MOVP T,@X2 ; MPYS P,T,@C2 ; MOV T,@X1 ; MPYA P,T,@C1 ; MOV T,@X0 ; MPYA P,T,@C0 ; ADDL ACC,P << PM ; MOVL @Y,ACC ; |

MPYB ACC,T,#8bit

Multiply by 8-bit Constant

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---|---|---|-----------------|---------|-------|--|--|
| MPYB ACC,T,#8bit | | | 0011 0101 CCCC CCCC | Х | - | 1 | | |
| Operands | ACC | Accumulator registe | r | | | | | |
| | т | Multiplicand register | | | | | | |
| | #8bit | 8-bit immediate constant value | | | | | | |
| Description | | Multiply the signed 16-bit content of the T register by the unsigned 8-bit constant value zero extended and store the result in the ACC register: | | | | | | |
| | | ACC = signed T $*$ | 0:8bit | | | | | |
| Flags and Modes | Z | After the operation, | the Z flag is set if the ACC i | is zero, else Z | is clea | ared. | | |
| | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Calcu ; Y32 = MOV MPYB ADDL | llate signed using Y32 + (X16 * 5) T,@X16 ; ACC,T,#5 ; @Y32,ACC ; | 16-bit multiply: T = X16 ACC = T * 5 Y32 = Y32 + ACC | | | | | |

MPYB P,T,#8bit

Multiply Signed Value by Unsigned 8-bit Constant

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--------------------------------|--|--|---------|-----|-----|--|--|--|
| MPYB P,T,#8bit | | | 0011 0001 CCCC CCCC | Х | - | 1 | | | |
| Operands | Р | Product register | | | | | | | |
| | т | Multiplicand register | | | | | | | |
| | #8bit | 8-bit immediate con | diate constant value | | | | | | |
| Description | | Multiply the signed 16-bit content of the T register by the unsigned 8-bit immediate constant value zero extended and store the 32-bit result in the P register: | | | | | | | |
| | | P = signed T * 0 | :8bit; | | | | | | |
| Flags and Modes | | None | | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Calcu MOV MPYB MOVL | late: Y32 = X16 * T,@X16 ; P,T,#5 ; @Y,P ; | 5; T = X16 P = T * #5 Store result into Y32 | | | | | | |

MPYS P,T,loc16

16 X 16-bit Multiply and Subtract

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-------|--|--|---|--|---|
| MPYS P,T,loc16 | | | 0001 0011 LLLL LLLL | Х | Y | N+1 |
| Operands | Р | Product register | | | | |
| | т | Multiplicand register | r | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Subtract the previous product (stored in the P register), shifted as specified by the product shift mode (PM), from the ACC register. In addition, multiply the signed 16-bit content of the T register by the signed 16-bit constant value and store the result in the P register: ACC = ACC - P << PM; P = signed T * signed [loc16]: | | | | |
| Flags and Modes | z | After the operation, | the Z flag is set if the ACC | is zero, else Z | is clea | ared. |
| | Ν | After the subtraction cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | e N is |
| | С | If the subtraction ge | nerates a carry, C is set; ot | herwise C is c | leared | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | |
| | OVC | If overflow mode is overflow, then the co the operation gen decremented. | disabled; and if the oper punter is incremented. If over erates a negative overfl | ation generat flow mode is c ow, then the | es a p lisablec e cour | ositive I; and if Iter is |
| | ΟνΜ | If overflow mode by positive (0x7FFFFF) overflowed. | it is set; then the ACC va FF) or maximum negative (0 | lue will satura x80000000) if | ate ma the op | ximum eration |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the the product shift value is a low bits are zero filled. If right shift operation), the up | he output oper positive (log f the product oper bits are s | ation fr gical le shift va ign exte | om the ft shift alue is ended. |
| Repeat | | This instruction is re it will be executed N reflect the final resu occurs. | peatable. If the operation foll I+1 times. The state of the ult. The V flag will be set i | lows a RPT in Z, N, C and (f an intermed | structio DVC fla liate ov | n, then ıgs will /erflow |

| <pre>Example ; Calculate using 16-bit multiply: ; Y = (X0*C0) >> 2) + (X1*C1 >> 2) + (X2*</pre> | | | | | |
|---|--|--|---|--|--|
| SPM | -2 | ; | Set product shift to >> 2 | | |
| MOVP | T,@X2 | ; | ACC = P, T = X2 | | |
| MPYS | P,T,@C2 | ; | ACC = ACC - P = 0, P = T*C2 | | |
| MOV | T,@X1 | ; | T = X1 | | |
| MPYA | P,T,@C1 | ; | ACC = $X2*C2>>2$, P = T*C1 | | |
| MOV | Τ,@ΧΟ | ; | T = X0 | | |
| MPYA | P,T,@C0 | ; | ACC = X1*C1>>2 + X2*C2>>2, P = T*C0 | | |
| ADDL | ACC,P << PM | ; | ACC = X0*C0>>2 + X1*C1>>2 + X2*C2>>2 | | |
| MOVL | @Y,ACC | ; | Store result into Y | | |
| | ; Calcu ; Y = SPM MOVP MPYS MOV MPYA ADDL MOVL | ; Calculate using 16-b ; Y = (X0*C0) >> 2) + SPM -2 MOVP T,@X2 MPYS P,T,@C2 MOV T,@X1 MPYA P,T,@C1 MOV T,@X0 MPYA P,T,@C0 ADDL ACC,P << PM MOVL @Y,ACC | ; Calculate using 16-bit ; Y = (X0*C0) >> 2) + (X SPM -2 ; MOVP T,@X2 ; MPYS P,T,@C2 ; MOV T,@X1 ; MPYA P,T,@C1 ; MOV T,@X0 ; MPYA P,T,@C0 ; ADDL ACC,P << PM ; MOVL @Y,ACC ; | | |

MPYU P,T,loc16

Unsigned 16 X 16 Multiply

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--------------------------------|---|---|---------------------------------|---------------------|-------------------|--|--|--|
| MPYU P,T,loc16 | | | 0011 0111 LLLL LLLL | Х | - | 1 | | | |
| Operands | P | Product register | | | | | | | |
| | I | Multiplicand register | | | | | | | |
| | loc16 | Addressing mode (s | Addressing mode (see Chapter 5) | | | | | | |
| Description | | Multiply the signed contents of the locat the 32-bit result in th P = unsigned T * | 16-bit content of the T region pointed to by the "loc16" ne P register: unsigned [loc16]; | gister by the addressing m | signed ode an | 16-bit d store | | | |
| Flags and Modes | | None | | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | → RPT nce. | | | |
| Example | ; Calcu MOV MPYU MOVL | late unsigned val T,@X16 ; P,T,@M16 ; @Y,P ; | ue: Y32 = X16 * M16; T = X16 P = T * M16 Store result into Y32 | | | | | | |
MPYU ACC,T,loc16

16 X 16-bit Unsigned Multiply

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|---|--|---|---------|-----|-----|--|--|--|
| MPYU ACC,T,I | oc16 | | 0011 0110 LLLL LLLL | Х | - | 1 | | | |
| Operands | ACC | Accumulator register | Accumulator register | | | | | | |
| | т | Multiplicand register | | | | | | | |
| | loc16 | Addressing mode (se | ee Chapter 5) | | | | | | |
| Description | | Multiply the unsigned 16-bit content of the T register by the unsigned 16-bit content of the location pointed to by the "loc16" addressing mode and stor the 32-bit results in the ACC register: | | | | | | | |
| | | ACC = unsigned T | * unsigned [loc16]; | | | | | | |
| Flags and Modes | Z | After the operation, the operation of th | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. | | | | | | |
| | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Calcu ; Y32 = MOV MPYU ADDL | ulate unsigned usin = Y32 + X16*M16 T,@X16 ; ACC,T,@M16 ; @Y32,ACC ; | ng 16-bit multiply: T = X16 ACC = T * M16 Y32 = Y32 + ACC | | | | | | |

MPYXU ACC, T, loc16

Multiply Signed Value by Unsigned Value

| SYNTAX OPTIONS | | | | OBJMODE | RPT | CYC | | |
|--------------------|--|---|-----------------------|---------|-----|-------|--|--|
| MPYXU ACC, | T, loc16 | | 0011 0000 LLLL LLLL | Х | - | 1 | | |
| Operands | ACC | Accumulator registe | Accumulator register | | | | | |
| | т | Multiplicand register | Multiplicand register | | | | | |
| | loc16 | Addressing mode (see Chapter 5) | | | | | | |
| Description | | Multiply the signed 16-bit content of the T register by the unsigned 16-bit content of the location pointed to by the "loc16" addressing mode and stort the result in the ACC register: | | | | | | |
| | | ACC = signed T * unsigned [loc16]; | | | | | | |
| Flags and Modes | z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. | | | | ared. | | |
| | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Calcu ; Y32 = MOV MPYXU ADDL | <pre>ulate signed using 16-bit multiply: = Y32 + (signed) X16 * (unsigned) M16 T,@X16 ; T = X16 ACC,T,@M16 ; ACC = T * M16 @Y32,ACC ; Y32 = Y32 + ACC</pre> | | | | | | |

MPYXU P,T,loc16

Multiply Signed Value by Unsigned Value

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|---|--|--|------|-----|--|
| MPYXU P,T,Io | c16 | | 0011 0010 LLLL LLLL | Х | - | 1 | |
| Operands | Р | Product register | | | | | |
| | т | Multiplicand register | | | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | | |
| | | | | | | | |
| Description | | Multiply the signed 16-bit content of the T register by the signed 16-bit contents of the location pointed to by the "loc16" addressing mode and store the 32-bit result in the P register: | | | | | |
| | | P = signed T * un | nsigned [loc16]; | | | | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; Calc MOV MPYU MOV MOVU MOV MPYXU | ulate "Y32 = X32 * T,@X32+0 ; ACC,T,@M32+0 ; @Y32+0,AL ; ACC,@AH ; T,@X32+1 ; P,T,@M32+0 ; | M32" by parts using 16 T = unsigned low X32 ACC = T * unsigned low Store low result into Logical shift right AC T = signed high X32 ACC = T * low unsigned | 5-bit multij M32 Y32 C by 16 M32 | oly: | | |
| | MOVA MPYXU ADDL MOV | T,@M32+1 ; P,T,@X32+0 ; ACC,@P ; @Y32+1,AL ; | T = Signed high M32, A ACC = T * low unsigned Add P to ACC Store high result into | CC += P X32 Y32 | | | |

Unalign Stack Pointer

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-----------------------------|---|---|--|----------------------------------|---------------------------|
| NASP | | | 0111 0110 0001 0111 | Х | - | 1 |
| Operands | | None | | | | |
| Description | | If the SPA bit is 1, the 1 and then clears the performed earlier by instruction performs | e NASP instruction decreme e SPA status bit. This undo the ASP instruction. If the no operation. | nts the stack p es a stack poir SPA bit is 0, th | ointer (nter alig nen the | (SP) by gnment NASP |
| | | if(SPA = 1) { SP = SP - 1; SPA = 0; } | | | | |
| Flags and Modes | PSA | If (SPA = 1), then SF | PA is cleared. | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT once. |
| Example | ; Alig: ; Vecto INTx: | nment of stack p or table: .long INTxService | oointer in interrupt e ; INTx interrup | service ro t vector | outine | 9: |
| | | | | | | |
| | INTxSer ASP | vice: | ; Align stack p | ointer | | |
| | NAS | P | ; Re-align stac | k pointer | | |
| | IRE | T | ; Return from i | nterrupt. | | |

NEG ACC

Negate Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| NEG ACC | 1111 1111 0101 0100 | Х | - | 1 |

| Operands | ACC | Accumulator register |
|--------------------|--|--|
| Description | | Negate the contents of the ACC register: |
| | | <pre>if(ACC = 0x8000 0000) { V = 1; if(OVM = 1) ACC = 0x7FFF FFFF; else ACC = 0x8000 0000; } else ACC = -ACC; if(ACC = 0x0000 0000) C = 1; else C = 0;</pre> |
| Flags and Modes | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | Z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. |
| | С | If (ACC = 0), set C; otherwise, clear C. |
| | V | If (ACC = $0x8000\ 0000$) at the start of the operation, this is considered an overflow value and V is set. Otherwise, V is not affected. |
| | ΟνΜ | If (ACC = $0x8000\ 0000$) at the start of the operation, this is considered an overflow value, and the ACC value after the operation depends on the state of OVM: If OVM is cleared, ACC will be filled with $0x8000\ 0000$. If OVM is set ACC will be saturated to $0x7FFF$ FFFF. |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |
| Example | ; Negat MOVL SETC NEG MOVL | e contents of VarA, make sure value is saturated: ACC,@VarA ; Load ACC with contents of VarA OVM ; Turn overflow mode on ACC ; Negate ACC and saturate @VarA,ACC ; Store result into VarA |

NEG AX

Negate AX Register

| s | SYNTAX (| OPTIONS | OPCODE | | | OBJMODE | RPT | CYC | |
|--------------------|--|---|-----------------------|-----------------|------------------|------------------|------------------|---------|---------|
| NEG AX | | | 1111 | 1111 | 0101 | 110A | Х | - | 1 |
| Operands | АХ | Accumulator high (AH) or accumulator low (AL) register | | | | | | | |
| Description | | Replace the contents of the specified AX register with the negative of AX: | | | | | | | of AX: |
| | | <pre>if(AX = 0x8000) { AX = 0x8000; V flag = 1; } else AX = -AX; if(AX = 0x0000) C flag = 1; else C flag = 0;</pre> | | | | | | | |
| Flags and Modes | Ν | After the operation, i otherwise, it is cleare | if bit 18 ed. | 5 of A | X is | 1, then t | he negative f | lag bit | is set; |
| | z | After the operation, if | f AX is (| 0, ther | n the Z | Z bit is se | t, otherwise it | is clea | ared. |
| | С | If AX is 0, C is set; ot | herwise | e, it is | cleare | ed. | | | |
| | V | If AX is 0x8000 at the and V is set. Otherw | start of vise V is | the op not a | eratio ffecte | n, then th d. | nis is considere | ed an o | verflow |
| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | | |
| Example | <pre>; Take the absolute value of VarA: MOV AL,@VarA ; Load AL with contents of VarA NEG AL ; If Al = 8000h, then V = 1 SB NoOverflow,NOV ; Branch and save -AL if no overflow MOV @VarA,0x7FFFh ; Save 7FFF if overflow NoOverflow: MOV @VarA,AL ; Save NEG AL if no overflow</pre> | | | | | | | w | |

NEG64 ACC:P

SYNTAX OPTIONS OPCODE OBJMODE RPT CYC NEG64 ACC:P 1 0101 0110 0101 1000 1 _ **Operands** ACC:P Accumulator register (ACC) and product register (P) Description Negate the 64-bit content of the combined ACC:P registers: if(ACC:P = 0x8000 0000 0000 0000){ $\dot{V} = 1;$ if(OVM = 1)ACC:P = 0x7FFF FFFF FFFF FFFF; else ACC:P = 0x8000 0000 0000;} else ACC:P = -ACC:P;if(ACC:P = 0x0000 0000 0000 0000)C = 1;else C = 0;Flags and Ν After the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and the Mode N bit is set; otherwise N is cleared. Ζ After the operation, the Z flag is set if the combined 64-bit value of the ACC:P is zero; otherwise, Z is cleared. С If (ACC:P = = 0) then the C bit is set; otherwise C is cleared. if(ACC:P = 0x8000 0000 0000 0000) then the V flag is set; otherwise, V is not ν modified. OVM If at the start of the operation, $ACC:P = 0x8000\ 0000\ 0000\ 0000$, then this is considered an overflow value and the ACC:P value after the operation depends on OVM. If (OVM = 1) ACC:P is filled with its greatest positive number (0x7FFF FFFF FFFF FFFF). If (OVM = 0) then ACC:P is not modified. Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Negate Accumulator Register and Product Register

| Example | ; Negate | e the contents of th | ie | 64-bit Var64 and saturate: |
|---------|----------|----------------------|----|-------------------------------------|
| | MOVL A | CC,@Var64+2 | ; | Load ACC with high 32-bits of Var64 |
| | MOVL P | ,@Var64+0 | ; | Load P with low 32-bits of Var64 |
| | SETC O | VM | ; | Enable overflow mode (saturate) |
| | NEG64 AG | CC:P | ; | Negate ACC:P with saturation |
| | MOVL @V | Var64+2,ACC | ; | Store high 32-bit result into Var64 |
| | MOVL @ | Var64+0,P | ; | Store low 32-bit result into Var64 |

NEGTC ACC

If TC is Equivalent to 1, Negate ACC

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| NEGTC ACC | 0101 0110 0011 0010 | 1 | - | 1 |

| Operands | ACC | Accumulator register |
|--------------------|-----|--|
| Description | | Based on the state of the test control (TC) bit, conditionally replace the content of the ACC register with its negative: |
| | | <pre>if(TC = 1) { if(ACC = 0x8000 0000) { V = 1; if(OVM = 1) ACC = 0x7FFF FFFF; else ACC = 0x8000 0000 } else ACC = 0x8000 0000</pre> |
| Flags and Modes | N | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | Z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. |
| | С | If (TC = 1 AND ACC = 0) set C; if (TC = 1 AND ACC != 0) clear C; otherwise C is not modified. |
| | V | If (TC = 1 AND ACC = $0x8000\ 0000$) at the start of the operation, this is considered an overflow value and V is set. Otherwise, V is not affected. |
| | тс | The state of the TC bit is used as a test condition for the operation. |
| | OVM | If at the start of the operation, ACC = $0x8000\ 0000$, then this is considered an overflow value and the ACC value after the operation depends on OVM. If OVM is cleared and TC = 1, ACC will be filled with 0x8000\ 0000. If OVM is set and TC = 1, ACC will be saturated to 0x7FFF FFFF. |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |

| Example | ; Calcula | te signed: Quot16 = | Num16/Den16, Rem16 = Num16%Den16 |
|---------|-----------|---------------------|------------------------------------|
| | CLRC | TC | ; Clear TC flag, used as sign flag |
| | MOV | ACC,@Den16 << 16 | ; $AH = Den16$, $AL = 0$ |
| | ABSTC | ACC | ; Take abs value, TC = sign ^ TC |
| | MOV | Т,@АН | ; Temp save Den16 in T register |
| | MOV | ACC,@Num16 << 16 | ; AH = Num16, AL = 0 |
| | ABSTC | ACC | ; Take abs value, TC = sign ^ TC |
| | MOVU | ACC,@AH | ; $AH = 0$, $AL = Num16$ |
| | RPT | #15 | ; Repeat operation 16 times |
| | SUBCU | @T | ; Conditional subtract with Den16 |
| | MOV | @Rem16,AH | ; Store remainder in Rem16 |
| | MOV | ACC,@AL << 16 | ; AH = Quot16, AL = 0 |
| | NEGTC | ACC | ; Negate if TC = 1 |
| | MOV | @Quot16,AH | ; Store quotient in Quot16 |

| NOP {*ind}{ | ARPn} | Operation With Optional Inc | direct Address | s Modifi | ication | |
|--------------------|---|--|---|---|----------------------|---------------------|
| | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | СҮС |
| NOP {*ind}. | ARPn} | | 0111 0111 LLLL LLLL | X | Y | N+1 |
| | | | | | | |
| Operands | {*ind} | Indirect address mo | de (see chapter 5) | | | |
| | ARPn | Auxiliary register po | inter (ARP0 to ARP7) | | | |
| Description | | Modify the indirect a register pointer (AR given, then do nothi | address operand as specific P) to the given auxiliary re ng. | ed and chang egister. If no | e the a operar | uxiliary ıds are |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is rep will execute N+1 tim | peatable. If this instruction for each of the second second second second second second second second second se | ollows the RP ⁻ | Г instru | ction, it |
| Example | MOVL MOVL MOV NOP SETC LOOP: MOVL NOP PDT | <pre>; Copy the conter ; int32 Array1[N] ; int32 Array2[N] ; for(i=0; i < N] ; Array2[i] = Arr ; This example on ; of program space XAR2,#Array1 XAR3,#Array2 @AR0,#(N-1) *,ARP2 AMODE ACC,* *++,ARP3 #19</pre> | <pre>hts of Array1 to Array2 ; ; ; i++) ray1[i]; hly works for code loca ce: ; XAR2 = pointer to A ; XAR3 = pointer to A ; Repeat loop N times ; Point to XAR2 (ARP ; Full C2xLP address ; ACC = Array1[i] ; Increment XAR2 and ; Do nothing for 20 or </pre> | 2: ated in upper array1 array2 = 2) mode compat point to XA | er 64K ible R3 | |
| | NOP MOVL | *++, ACC, ARP0 | ; Array2[i] = ACC, pc | oint to XAR |) | |
| | XBANZ | Loop,*,ARP2 | ; Loop if AR[ARP] != ; point to XAR2 | 0, AR[ARP]- | -, | |

| NORM ACC, *ind Normalize ACC and Modify Selected Auxiliary Reg | | | | egister | | | | |
|--|----------|--|--|--|--|--|--|--|
| | SYNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
| NORM ACC, * | | | 0101 0110 0010 0100 | 1 | Y | N+4 | | |
| NORM ACC, * | ++ | | 0101 0110 0101 1010 | 1 | Y | N+4 | | |
| NORM ACC, * | | | 0101 0110 0010 0000 | 1 | Y | N+4 | | |
| NORM ACC, * | 0++ | | 0101 0110 0111 0111 | 1 | Y | N+4 | | |
| NORM ACC, * | 0 | | 0101 0110 0011 0000 | 1 | Y | N+4 | | |
| Operands | ACC | Accumulator registe | er | | | | | |
| | *ind | *, *++, *, *0++, *0 | 0 indirect addressing moc | les (see Chap | oter 5) | | | |
| Description | | Normalize the signe the indirect address to by the auxiliary re | ed content of the ACC registe ing mode, the auxiliary regis egister pointer (ARP): | r and modify, a ter (XAR0 to > | as spec (AR7) p | ified by pointed | | |
| | | Note: The NORM inst magnitude of th bits are the san eliminate the ex the ACC is not does not acces | truction normalizes a signed numbe e number. An XOR operation is perfor- ne, then the content of the ACC reg tra sign bit and the selected pointer shifted and the selected pointer is s any memory location. | r in the ACC regis ormed on ACC bits jister is logically s is modified. If the not modified. The | ster by fir s 31 and 3 shifted let bits are selected | nding the 30. If the ft by 1 to different, d pointer | | |
| Flags and | Z | After the operation, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | | | |
| Modes | Ν | After the operation, | the N flag is set if bit 31 of the | ACC is 1, els | e N is c | leared. | | |
| | тС | If the operation set be modified). If the o as a result, the ACO | If the operation set TC, no normalization was needed (ACC did not need to be modified). If the operation cleared TC, bits 31 and 30 were the same and, as a result, the ACC register was logically shifted left by 1. | | | | | |
| | ARP | Auxiliary register p operation (XAR0 to | ointer selects which pointe XAR7). | r to modify a | ıs part | of the | | |
| Repeat | | This instruction is re the NORM instruction TC flags will reflect instruction to execu checks the value of | peatable. If the operation foll on will be executed N+1 time of the final result. Note: If te until normalization is done the TC bit. When TC = 1, r | ows a RPT in s. The state o you only war e, you can crea normalization | structio f the Z, nt the ate a lo is com | n, then N, and NORM op that plete. | | |

| Example | ; Normali ; XAR2 wi | ze the conte ll contain s | nts of VarA, nift value at the end of the operation: | |
|---------|------------------------|------------------------------|---|----|
| | MOVL | ACC,@VarA | ; ACC = VarA | |
| | MOVB | XAR2,#0 | ; Initialize XAR2 to zero | |
| | NOP | *,ARP2 | ; Set ARP pointer to point to XAR | 22 |
| | SBF | Skip,EQ | ; Skip if ACC value is zero | |
| | RPT | #31 | ; Repeat next operation 32 times | |
| | NORM | ACC, *++ | ; Normalize contents of ACC | |
| | Skip: | | | |

NORM ACC,XARn++/--

Normalize ACC and Modify Selected Auxiliary Register.

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------------------|---------------------|---------|-----|-----|
| NORM ACC,XARn++ | 1111 1111 0111 1nnn | Х | Y | N+4 |
| NORM ACC,XARn | 1111 1111 0111 Onnn | Х | Y | N+4 |
| Operands ACC Accumulator regis | tor | | | |

| Operands | ACC | Accumulator register |
|--------------------|-------------|---|
| | XARn ++/ | XAR0 to XAR7, auxiliary registers post incremented or decremented |
| Description | | Normalize the signed content of the ACC register and modify the specified auxiliary register (XAR0 to XAR7): |
| | | if(ACC != 0x0000 0000) |
| | | { if((ACC(31) XOR ACC(30)) = 0) $($ |
| | | { ACC = ACC << 1, TC = 0; |
| | | if (XARn++ addressing mode) XARn += 1; |
| | | <pre>if(XARn addressing mode) XARn -= 1; }</pre> |
| | | else |
| | | } |
| | | else TC - 1. |
| | | Note: The NORM instruction normalizes a signed number in the ACC register by finding the |
| | | magnitude of the number. An XOR operation is performed on ACC bits 31 and 30. If the bits are the same, then the content of the ACC register is logically shifted left by 1 to eliminate the extra sign bit and the selected pointer is modified. If the bits are different, the ACC is not shifted and the selected pointer is not modified. The selected pointer does not access any memory location. |
| Flags and Modes | z | After the operation, the Z flag is set if the ACC value is zero, else Z is cleared. |
| | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
| | тс | If the operation set TC, no normalization was needed (ACC did not need to be modified). If the operation cleared TC, bits 31 and 30 were the same and, as a result, the ACC register was logically shifted left by 1. |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then the NORM instruction will be executed N+1 times. The state of the Z, N, and TC flags will reflect the final result. Note: If you only want the NORM instruction to execute until normalization is done, you can create a loop that checks the value of the TC bit. When TC = 1, normalization is complete. |

| Example | ; Norma ; XAR2 | alize the contents will contain shift | of VarA, t value at the end of the operation: |
|---------|-------------------|--|--|
| | MOVL | ACC,@VarA | ; ACC = VarA |
| | MOVB | XAR2,#0 | ; Initialize XAR2 to zero |
| | SBF | Skip,EQ | ; Skip if ACC value is zero |
| | RPT | #31 | ; Repeat next operation 32 times |
| | NORM | ACC, XAR2++ | ; Normalize contents of ACC |
| | Skip: | | |

NOT ACC Complement Accumulator SYNTAX OPTIONS OPCODE OBJMODE RPT CYC NOT ACC Х 1111 1111 0101 0101 1 _ Operands ACC Accumulator register Description The content of the ACC register is replaced with its complement: ACC = ACC XOR 0xFFFFFFF; Flags and Ν After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. Modes Ζ After the operation, the Z flag is set if the ACC is zero, else Z is cleared. Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. Example ; Complement the contents of VarA: MOVL ACC,@VarA ; ACC = VarA NOT ACC ; Complement ACC contents @VarA,ACC MOVL ; Store result into VarA

NOT AX

Complement AX Register

| SY | NTAX OF | TIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--|------------------------------------|---|---------------|----------|----------|--|--|--|
| NOT AX | | | 1111 1111 0101 111A | Х | - | 1 | | | |
| Operands | АХ | Accumulator high | (AH) or accumulator low (AL | .) register | | | | | |
| Description | | Replace the cor complement: | ntents of the specified AX r | egister (AH c | or AL) v | with its | | | |
| | | AX = AX XOR 0x | FFFF; | | | | | | |
| Flags and Modes | Ν | After the operation it is cleared. | After the operation, if bit 15 of AX is 1 then the negative flag bit is set; otherwise it is cleared. | | | | | | |
| | Z | After the operatio | After the operation, if AX is 0, then the Z bit is set, otherwise it is cleared. | | | | | | |
| Repeat | Image: Provide and the image: This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | | |
| Example | ; Comp | lement the conte | ents of VarA: | | | | | | |
| • | MOV | AL,@VarA | ; Load AL with con | ntents of Va | rA | | | | |
| | NOT | AL | ; Complement conte | ents of AL | | | | | |
| | MOV | @VarA,AL | ; Store result in | VarA | | | | | |

OR ACC, loc16

| S | YNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|------------------------------|---|---|--|---------------------|--------------------|--|
| OR ACC, loc16 | | | 1010 1111 LLLL LLLL | 1 | Y | N+1 | |
| Operands | ACC loc16 | Accumulator registe Addressing mode (s | r ee Chapter 5) | | | | |
| Description | | Perform a bitwise Of content of the location stored in the ACC re ACC = ACC OR | R operation on the ACC region pointed to by the "loc16" agister: 0 : [loc16] ; | ster with the z address mode | ero-ext . The re | tended esult is | |
| Flags and Modes | N Z | The load to ACC is to the negative flag bit The load to ACC is t operation generates | The load to ACC is tested for a negative condition. If bit 31 of ACC is 1, then the negative flag bit is set; otherwise it is cleared. The load to ACC is tested for a zero condition. The zero flag bit is set if the operation generates $ACC = 0$; otherwise it is cleared | | | | |
| Repeat | | This operation is repeatable. If the operation follows a RPT instruction, then the OR instruction will be executed N+1 times. The state of the Z and N flags will reflect the final result. | | | | | |
| Example | ; Calc MOVL OR MOVL | ulate the 32-bit va ACC,@VarA ACC,@VarB @VarA,ACC | alue: VarA = VarA OR 0: ; Load ACC with conto ; OR ACC with content ; Store result in Va: | VarB ents of Var ts of 0:Var rA | A B | | |

OR ACC,#16bit << #0..16

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-----------------------|--|---------|-----|-----|
| OR ACC,#16bit << #015 | 0011 1110 0001 SHFT CCCC CCCC CCCC CCCC | 1 | I | 1 |
| OR ACC,#16bit << #16 | 0101 0110 0100 1010 CCCC CCCC CCCC CCCC | 1 | 1 | 1 |

| Operands | ACC #16bit #016 | Accumulator register 16-bit immediate constant value Shift value (default is "<< #0" if no value specified) |
|--------------------|-----------------------|---|
| Description | | Perform a bitwise OR operation on the ACC register with the given 16-bit unsigned constant value left shifted as specified. The value is zero extended and lower order bits are zero filled before the OR operation. The result is stored in the ACC register: ACC = ACC OR (0:16bit << shift value); |
| Flags and Modes | Ν | The load to ACC is tested for a negative condition. If bit 31 of ACC is 1, then the negative flag bit is set; otherwise it is cleared. |
| | Z | The load to ACC is tested for a zero condition. The zero flag bit is set if the operation generates $ACC = 0$; otherwise it is cleared |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |
| Example | ; Calcu | late the 32-bit value: VarA = VarA OR 0x08000000 |
| - | MOVL A | CC,@VarA ; Load ACC with contents of VarA |
| | OR A | CC,#0x8000 << 12 ; OR ACC with 0x08000000 |
| | MOVL @ | /arA,ACC ; Store result in VarA |

OR AX, loc16

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-----------------------------|--|--|--|-------------------|----------|
| OR AX, loc16 | | | 1100 101A LLLL LLLL | Х | - | 1 |
| Operands | AX loc16 | Accumulator high (A Addressing mode (s | H) or accumulator low (AL) ee Chapter 5) | register | | |
| Description | | Perform a bitwise Of contents of the locat result is stored in AX AX = AX OR [loc16 | R operation on the specified ion pointed to by the "loc16 K: 5] ; | d AX register " addressing | with the mode. | ∍ The |
| Flags and Modes | Ν | The load to AX is tes negative flag bit is se | ted for a negative condition. et; otherwise it is cleared. | If bit 15 of AX | (is 1, th | ien the |
| | Z | The load to AX is tested for a zero condition. The zero flag bit is set if the operation generates $AX = 0$, otherwise it is cleared. | | | | |
| Repeat | | This instruction is no instruction, it resets | ot repeatable. If this instruct the repeat counter (RPTC) | ion follows the and executes | e RPT s only o | nce. |
| Example | ; OR th MOV OR MOV | ne contents of Vari AL,@VarA AL,@VarB @VarC,AL | A and VarB and store in ; Load AL with cont ; OR AL with conter ; Store result in V | n VarC: cents of Var nts of VarB <i>V</i> arC | CA | |

OR IER,#16bit

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--------------------------|---|--|----------------------------------|----------------------|------------------|--|--|
| OR IER,#16bit | | | 0111 0110 0010 0011 CCCC CCCC CCCC CCCC | Х | I | 2 | | |
| Operands | IER | Interrupt enable regi | ister | | | | | |
| | #16bit- Mask | 16-bit immediate co | 16-bit immediate constant value | | | | | |
| Description | | Enable specific inter- register and the 16- register: | rupts by performing a bitwise -bit immediate value. The | e OR operatio result is store | n with t ed in th | he IER ne IER | | |
| | | IER = IER OR #1 | .6bit; | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. | | |
| Example | ; Enabl ; inter OR | e INT1 and INT6 or rupt's enable: IER,#0x0061 | nly. Do not modify stat ; Enable INT1 and INT | te of other 6 | | | | |

OR IFR,#16bit

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------------------------|---|--|--|------------------------------------|--------------------------------|
| OR IFR,#16bit | | | 0111 0110 0010 0111 CCCC CCCC CCCC CCCC | Х | - | 2 |
| Operands | IFR #16bit | Interrupt flag registe 16-bit immediate cor | r nstant value | | | |
| Description | | Enable specific intern register and the 16-1 stored in the IFR reg | rupts by performing a bitwise bit immediate value. The re gister. | e OR operatio sult of the Of | n with t R opera | he IFR ation is |
| | | IFR = IFR OR #16k | pit; | | | |
| | | Note: Interrupt hardwa interrupt flag is b This instruction shou interrupt expansion | are has priority over CPU instruct being simultaneously modified by the ld not be used with interrupt (PIE) block is enabled. | ion operation in he hardware and s 1–12 when | cases wh the instru the peri | here the uction. ipheral |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is no struction, it resets th | ot repeatable. If this instruct re repeat counter (RPTC) a | ion follows the nd executes o | e RPT i only ond | in- ce. |
| Example | ; Trigg ; inter OR | er INT1 and INT6 o rupt's flags: IFR,#0x0061 | only. Do not modify sta ; Trigger INT1 and IN | ate of other T6 | 2 | |

OR loc16,#16bit

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|----------------|---------------------------|---|--|--|---------------------|---------------------|--|--|--|
| OR loc16,#16bi | it | | 0001 1010 LLLL LLLL | Х | - | 1 | | | |
| | | | CCCC CCCC CCCC | | | | | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | | | | |
| | #16bit | 16-bit immediate cor | nstant value | | | | | | |
| Description | | Perform a bitwise OI the "loc16" addressin result is stored in the | rform a bitwise OR operation on the content of the location pointed to by "loc16" addressing mode and the 16-bit immediate constant value. The sult is stored in the location pointed to by "loc16": | | | | | | |
| | | [loc16] = [loc16] OR | 16bit; | | | | | | |
| | | Smart Encoding: If loc16 = AH or AL encode this instruction this encoding, use the | and #16bit is an 8-bit numb on as ORB AX, #8bit to imp ne ORW AX, #16bit instruct | per, then the a prove efficienc ion alias. | assemb y. To ov | ler will /erride | | | |
| Flags and | Ν | After the operation if | f bit 15 of [loc16] 1, set N; o | therwise, clea | ar N. | | | | |
| Modes | Z | After the operation if | f [loc16] is zero, set Z; othe | rwise, clear Z | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. | | | |
| Example | ; Set E ; VarA OR @ | Sits 4 and 7 of Va: = VarA OR #(1 << 4 VarA,#(1 << 4 1 | rA: 4 1 << 7) << 7) ; Set bits | 4 and 7 of | VarA | | | | |

OR loc16, AX

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-------------|--|--|-------------------------------------|----------------------|------------------|
| OR loc16, AX | | | 1001 100A LLLL LLLL | X | - | 1 |
| Operands | loc16 AX | Addressing mode (s Aaccumulator high (| ee Chapter 5) AH) or accumulator low (Al | _) register | | |
| Description | | Perform a bitwise Of "loc16" addressing n in the addressed loc | R operation on the contents node with the specified AX r ation specified by "loc16": | of location poi register. The re | inted to esult is | by the stored |
| | | [10010] = [10010] | OR AA, | | | |
| | | This instruction perfe | orms a read-modify-write op | peration. | | |
| Flags and Modes | N | The load to [loc16] is then the negative fla | s tested for a negative cond Ig bit is set; otherwise it is c | ition. If bit 15 (leared. | of [loc1 | 6] is 1, |
| | z | The load to [loc16] is operation generates | tested for a zero condition. [loc16] = 0, otherwise it is | The zero flag cleared. | bit is se | et if the |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |
| Example | ; OR th | ne contents of Vari | A with VarB and store : | in VarB: | | |
| | MOV | AL,@VarA | ; Load AL with cont | ents of Var | :A | |
| | OR | @VarB,AL | ; VarB = VarB OR AI | L | | |

ORB AX,#8bit

Bitwise OR 8-bit Value

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|------------------------------|--|--|-----------------------------------|-------------------|------|--|--|--|
| ORB AX, #8bit | | | 0101 000A CCCC CCCC X - 1 | | | | | | |
| Operands | AX #8bit | Accumulator high (AH) or accumulator low (AL) register 3-bit immediate constant value | | | | | | | |
| Description | | Perform a bitwise OI unsigned immediate AX = AX OR 0x00:8 | Perform a bitwise OR operation on the specified AX register with the 8-bit insigned immediate constant zero extended. The result is stored in AX: X = AX OR 0x00:8bit; | | | | | | |
| Flags and Modes | Ν | The load to AX is tes negative flag bit is se | ne load to AX is tested for a negative condition. If bit 15 of AX is 1, then the egative flag bit is set; otherwise it is cleared. | | | | | | |
| | z | The load to AX is tested for a zero condition. The zero flag bit is set if the operation generates $AX = 0$, otherwise it is cleared. | | | | | | | |
| Repeat | | This instruction is no instruction, it resets | ot repeatable. If this instruct the repeat counter (RPTC) | ion follows the and executes | e RPT s only o | nce. | | | |
| Example | ; Set k MOV ORB MOV | oit 7 of VarA and s AL,@VarA AL,#0x80 @VarB,AL | store result in VarB: ; Load AL with cont ; OR contents of AI ; Store result in V | ents of Var with 0x008 MarB | rA 30 | | | | |

OUT *(PA),loc16

Output Data to Port

| 5 | Ο ΧΑΤΛΥ | PTIONS | ; | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|--|--|--|---|---|--|---|
| OUT *(PA),loc1 | 6 | | | 1011 CCCC | 1100 LLLL LLLL CCCC CCCC CCCC | 1 | - | 4 |
| Operands | *(PA) loc16 | Imme Addre | diate I/O space ssing mode (s | e men see Ch | nory address napter 5) | | | |
| Description | | Store mode IOspa I/O Sp interfa | tore the 16-bit value from the location pointed to by the "loc16" addre node into the I/O space location pointed to by the *(PA) operand): Ospace [0x0000PA] = [loc16]; O Space is limited to 64K range (0x0000 to 0xFFFF). On the ext iterface (XINTF), if available on a particular device, the I/O strobe st XISn) is toggled during the operation. The I/O address appears on the | | | | | |
| | | 16 XII The d | NTF address line | nes (X n the I | (A(15:0)) and the upper ower 16 data lines (X | er address line D(15:0). | es are z | zeroed. |
| | | Note: | The UOUT operations of the UOUT operation of the | ation is instruc eration. | not pipeline protected. Hen tion, the IN will occur befo use the OUT instruction. w | ce, if an IN instru re the UOUT. To hich is pipeline p | ction imn be certa protected | nediately iin of the |
| | | Note: | The UOUT ope immediately follo certain of the s protected. I/O space may r particular device | eration ows a l equenc not be ir e for det | is not pipeline protected JOUT instruction, the IN w e of operation, use the C nplemented on all C28x de ails. | . Therefore, if a vill occur before DUT instruction, vices. See the da | an IN in the UOU which is ata sheet | struction T. To be pipeline t for your |
| Flags and Modes | | None | | | | | | |
| Repeat | | This i instrue | instruction is ction, it resets | not re | epeatable. If this in peat counter (RPTC) | struction follo and executes | ows the | e RPT once. |
| Example | ; IOReg ; IOREg ; IOReg ; IOReg ; IOReg ; IORegA IORegA IORegB IORegC MOV UOUT MOV UOUT OUT | gA addr gB addr gC addr gA = 02 gB = 02 gC = Va [ORegC (ORegC .set .set .set .set .set .set .set .set | <pre>cess = 0x030 cess = 0x030 cess = 0x030 c0000; arA; = 0x2000) = 0x0000; 0x0300 0x0301 0x0302 #0 DRegA),@AL #0x0400 DRegB),@AL DRegC),@VarA</pre> | 0; 1; 2; ; ; ; ; ; ; ; ; ; ; ; ; ; ; | Define IORegA add Define IORegB add Define IORegC add AL = 0 IOspace[IORegA] = AL = 0x0400 IOspace[IORegB] = IOspace[IORegC] = | lress lress lress = AL = AL = VarA | | |

| IN | <pre>@AL,*(IORegC)</pre> | ; AL = IOspace[IORegC] |
|-------|--------------------------|------------------------------|
| CMP | @AL,#0x2000 | ; Set flags on (AL - 0x2000) |
| SB | \$10,NEQ | ; Branch if not equal |
| MOV | @AL,#0 | ; AL = 0 |
| UOUT | *(IORegC),@AL | ; IOspace[IORegC] = AL |
| \$10: | | |

POP ACC

Pop Top of Stack to Accumulator

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|--|---|--|---------------------------------|----------------------------------|
| POP ACC | | | 0001 1110 1011 1110 | Х | - | 1 |
| Operands | ACC | Accumulator | | | | |
| Description | | Predecrement SP by | y 2. Load ACC with the 32-b | oit value point | ed to b | y SP: |
| | | SP -= 2; ACC = [SP]; | | | | |
| Flags and Modes | Ν | The load to ACC is to is the sign bit, 0 for p operation on the AC cleared. | ested for a negative condition positive, 1 for negative. The C register generates a neg | n. Bit-31 of the negative flag gative value, o | e ACC r bit is se otherwi | egister et if the se it is |
| | Z | The load to ACC is to operation on the AC | ested for a zero condition. Th C register generates a 0 va | e bit is set if th lue, otherwise | ne resul e it is cl | t of the leared |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo | ws the only o | e RPT nce. |

POP ARn:ARm

Pop Top of Stack to 16-bit Auxiliary Registers

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| POP AR1:AR0 | 0111 0110 0000 0111 | Х | - | 1 |
| POP AR3:AR2 | 0111 0110 0000 0101 | Х | _ | 1 |
| POP AR5:AR4 | 0111 0110 0000 0110 | Х | - | 1 |

| Operands | ARn: | AR1:AR0 or AR3:AR2 or AR5:AR4 auxiliary registers |
|----------|------|---|
| - | ARm | |

Description AR1:AR0 or AR3:AR2 or AR5:AR4 Predecrement SP by 2. Load the contents of two 16-bit auxiliary registers (ARn and ARm)with the value pointed to by SP and SP+1.

| <pre>POP AR1:AR0 SP -= 2; AR0 = [SP]; AR1 = [SP+1]; AR1H:AR0H = unchanged;</pre> |
|--|
| <pre>POP AR3:AR2 SP -= 2; AR2 = [SP]; AR3 = [SP+1]; AR3H:AR2H = unchanged;</pre> |
| <pre>POP AR5:AR4 SP -= 2; AR4 = [SP]; AR5 = [SP+1]; AR5H:AR4H = unchanged;</pre> |

Flags and Modes None

RepeatThis instruction is not repeatable. If this instruction follows the RPT
instruction, it resets the repeat counter (RPTC) and executes only once.

| POP AR1H:A | ROH | | Рор Т | op of Sta | ack to Uppe | er Half of Auxili | ary Reg | gisters |
|--------------------|---------------|--|-----------------------------------|--------------------------------------|---|---|----------------------|----------------------|
| | SYNTAX O | PTIONS | | OPCC | DE | OBJMODE | RPT | CYC |
| POP AR1H:AF | ROH | | 0000 | 0000 0 | 000 0011 | Х | - | 1 |
| Operands | AR1H: AR0H | Upper 16-bits of XA | AR1 an | d XAR0 a | auxiliary reç | gisters | - | |
| Description | | Predecrement SP by SP and AR1H w auxiliary registers (| by 2. Lo rith the v (AR0 ar | ad the co value poir nd AR1) a | ontents of A nted to by S are left uncl | R0H with the va P+1. The lowe nanged. | alue poi r 16 bit | inted to s of the |
| | | SP -= 2; AR0H = [SP]; AR1H = [SP+1]; AR1:AR0 = unchar | nged; | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is instruction, it resets | s not re s the re | epeatable peat cou | e. If this in nter (RPTC | nstruction follo | ows the | e RPT once. |
| Example | | | ; | Full co | ontext rea | store for an | | |
| • | • | | ; | interru | upt or tra | ap function | | |
| | POP | ХT | | 32-bit | XT resto | ^e | | |
| | POP | XAR7 | ; | 32-bit | XAR7 rest | core | | |
| | POP | XAR6 | ; | 32-bit | XAR6 rest | core | | |
| | POP | XAR5 | ; | 32-bit | XAR5 rest | core | | |
| | POP | XAR4 | ; | 32-bit | XAR4 rest | core | | |
| | POP | XAR3 | ; | 32-bit | XAR5 rest | core | | |
| | POP | XAR2 | ; | 32-bit | XAR2 rest | core | | |
| | POP | AR1H:AR0H | ; | 16-bit | AR1H and | 16-bit AROH | resto | re |
| | IRE | Т | | | | | | |

POP DBGIER

Pop Top of Stack to DBGIER

| POP DBGIER 0111 0110 0001 0010 | Х | 1 | 5 |
|--------------------------------|---|---|---|

| Operands | DBGIER | Debug interrupt-enable register |
|--------------------|--------|--|
| Description | | Predecrement SP by 1. Load the contents of DBGIER with the value pointed to by SP: |
| | | SP -= 1; DBGIER = [SP]; |
| Flags and Modes | | None |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |

POP DP

Pop Top of Stack to the Data Page

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| POP DP | 0111 0110 0000 0011 | Х | Ι | 1 |

| Operands | DP | Data-page register |
|--------------------|----|--|
| Description | | Predecrement SP by 1. Load the contents of DP with the value pointed to by SP: |
| | | SP -= 1; DP = [SP]; |
| Flags and Modes | | None |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |

POP DP:ST1

Pop Top of Stack to DP and ST1

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|------------|--|---|---------------------------------|-------------------|-------------|
| POP DP:ST1 | | | 0111 0110 0000 0001 | Х | - | 5 |
| Operands | DP:S T1 | data page register ar | nd status register 1 | | | |
| Description | | Predecrement SP by with the value pointer | 2. Load ST1 with the value p d to by SP+1: | ointed to by S | P and lo | oad DP |
| | | SP -= 2; ST1 = [SP]; DP = [SP+1]; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets t | not repeatable. If this ins he repeat counter (RPTC) a | struction follo and executes | ws the only or | PPT nce. |

| POP IFR Pop Top of Stack to | | | | to IFR | |
|-----------------------------|--|--|---------------------------------|---------------------|----------------|
| SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
| POP IFR | | 0000 0000 0000 0010 | Х | - | 5 |
| Operands IFR | Interrupt flag registe | Ðr | - | - | - |
| Description | Predecrement SP b SP: | y 1. Load the contents of IFF | ? with the value | e pointe | ed to by |
| | SP -= 1; IFR = [SP]; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT once. |

POP loc16

Pop Top of Stack

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---------------------------|---|--|--|------------------------------|-------------------------------|
| POP loc16 | 6 0010 1010 LLLL LLLL X - | | | 2 | | |
| Operands | loc16 | Addressing mode (S | See Chapter 5) | | | |
| Description | | Predecrement SP b pointed to by SP. | y 1. Load the contents of | loc16 with the | e 16-bit | t value |
| | | SP -= 1; [loc16] = [SP], | ; | | | |
| Flags and Modes | N | If (loc16 = @AX), the of the AX register is flag bit is set if the op otherwise it is cleare | en the load to AX is tested for the sign bit, 0 for positive, 1 peration on the AX register ed. | r a negative co I for negative. generates a n | ndition The ne egative | . Bit-15 egative value, |
| | z | If (loc16 = @AX), the set if the result of the otherwise it is cleare | en the load to AX is tested fo he operation on the AX reg ed | r a zero condi gister generat | tion. Th es a 0 | ie bit is value, |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT |
| Example | POP @T | | ; Predecrement SP } ; XT(31:15) with th ; contents of the 1 ; by SP TL is und | by 1. Load ne location poi | nted | to |
| | POP @AI | <u>.</u> | ; by SP. II IS uncl ; Predecrement SP I ; the contents of 1 ; to by SP. AH is a | by 1. Load A the location unchanged. | AL wit: n poin | h ted |
| | POP @AI | R4 | ; Predecrement SP } ; the contents of t ; to by SP. AR4H is | by 1. Load A the location s unchanged. | AR4 wi 1 poin | th ted |
| | POP *X | AR4++ | ; Predecrement SP B ; 16-bit location p ; with the contents ; pointed to by SP Post-increment ; XAR4 by 1 | by 1. Load to be a constructed to be constructed to be constructed to be a constructed | the by XAR cation | 4 |

| POP | Ρ |
|-----|---|
|-----|---|

Pop top of Stack to P

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|--|---------------------------------|------------------|---------------|--|--|
| POP P | | 0111 0110 0001 0001 | Х | - | 1 | | |
| Operands P | Product register | | | | | | |
| Description | Predecrement SP by | Predecrement SP by 2. Load P with the 32-bit value pointed to by SP: | | | | | |
| | SP -= 2; P = [SP]; | | | | | | |
| Flags and Modes | None | | | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. | | |
POP RPC

Pop RPC Register From Stack

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| POP RPC | 0000 0000 0000 0111 | Х | - | 3 |

| Operands | RPC | Return program counter register |
|--------------------|-----|--|
| Description | | Predecrement SP by 2. Load the contents of RPC with the value pointed to by SP: |
| | | SP -= 2; RPC = [SP]; |
| Flags and Modes | | None |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |

| POP ST0 | | | | Pop Top of | Stack t | o STO |
|--------------------|---------------------------------------|--|--|---------------------------------|------------------|---------------|
| | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
| POP ST0 | | | 0111 0110 0001 0011 | Х | _ | 1 |
| Operands | ST0 | status register 0 | | | | |
| Description | | Predecrement SP by SP: | / 1. Load the contents of STC |) with the value | e pointe | d to by |
| | | SP -= 1; ST0 = [SP]; | | | | |
| Flags and Modes | C N Z TC SXM OVC PM | The bit value of each of the stack | flag and mode listed is repla | aced by the va | lue pop | ped off |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this insthe repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |

| r | | | | | | |
|--------------------|---|--|--|---------------------------------|------------------|----------|
| s | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
| POP ST1 | | | 0111 0110 0000 0000 | Х | - | 5 |
| Operands | ST1 | Status register 1 | | | | |
| Description | | Predecrement SP by SP: | / 1. Load the contents of STC |) with the value | e pointe | d to by |
| | | SP -= 1; ST1 = [SP]; | | | | |
| Flags and Modes | DBGM INTM VMAP SPA PAGE0 AMODE ARP EAL- LOW OBJ- MODE XF | The bit values for ea off of the stack | ch flag and mode listed is re | placed by the | value p | iopped |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this insthe repeat counter (RPTC) | struction follo and executes | ws the only o | RPT nce. |

POP ST1

Pop Top of Stack to ST1

POP T:ST0

Pop Top of Stack to T and ST0

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---|--|---------------------------------|---------------------------------|-------------------|--|--|
| POP T:ST0 | | 0111 0110 0001 0101 | Х | - | 1 | | |
| Operands T:ST0 | The upper 16-bits of | The upper 16-bits of the multiplicand register and status register 0 | | | | | |
| Description | Predecrement SP by with the value pointer left unchanged: | y 2. Load ST0 with the value d to by SP+1. The low 16 bits | pointed to by s of the XT Re | SP and gister (⁻ | load T TL) are | | |
| | <pre>SP -= 2; T = [SP]; ST0 = [SP+1]; TL = unchanged;</pre> | | | | | | |
| Flags and Modes | None | | | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo | ws the only o | e RPT nce. | | |

| SYNTAX OPTIONS OPCODE OBJMOD POP XAR0 0011 1010 1011 1110 1 POP XAR1 1011 0010 1011 1110 1 POP XAR2 1010 1010 1011 1110 1 POP XAR3 1010 0010 1011 1110 1 POP XAR4 1010 0010 1011 1110 1 POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | E RPT | CYC 1 1 1 1 1 1 1 1 1 1 1 1 |
|--|--------------------------|---|
| POP XAR0 0011 1010 1011 1110 1 POP XAR1 1011 0010 1011 1110 1 POP XAR2 1010 1010 1011 1110 1 POP XAR3 1010 0010 1011 1110 1 POP XAR4 1010 1000 1011 1110 1 POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | | 1 1 1 1 1 1 1 1 1 1 |
| POP XAR1 1011 0010 1011 1110 1 POP XAR2 1010 1010 1011 1110 1 POP XAR3 1010 0010 1011 1110 1 POP XAR4 1010 1000 1011 1110 1 POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | - | 1 1 1 1 1 1 1 1 |
| POP XAR2 1010 1011 1110 1 POP XAR3 1010 0010 1011 1110 1 POP XAR4 1010 1000 1011 1110 1 POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1100 X | | 1 1 1 1 1 1 |
| POP XAR3 1010 0010 1011 1110 1 POP XAR4 1010 1000 1011 1110 1 POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | - | 1 1 1 1 1 |
| POP XAR4 1010 1010 1011 1110 1 POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | | 1 1 1 |
| POP XAR5 1010 0000 1011 1110 1 POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | | 1 1 1 |
| POP XAR6 1100 0010 1011 1110 X POP XAR7 1100 0011 1011 1110 X | | 1 |
| POP XAR7 1100 0011 1011 1110 X | | 1 |
| | | |
| SP -= 2; XARn = [SP]; Flags and None Modes | | - |
| Repeat This instruction is not repeatable. If this instruction for instruction, it resets the repeat counter (RPTC) and execution. Execution Formula | ollows th ites only o | ie RPT once. |
| Example . ; Full context restore for | an n | |
| . , interrupt of trap functio | | |
| POP XT ; 32-bit XT restore | | |
| POP XAR7 ; 32-bit XAR7 restore | | |
| POP XAR6 ; 32-bit XAR6 restore | | |
| POP XAR5 ; 32-bit XAR5 restore | | |
| POP XAR4 ; 32-bit XAR4 restore | | |
| PUP XAR3 ; 32-DIT XAR3 restore | | |
| FUF AARZ ; 32-DIL AARZ restore | OU roct | ore |

IRET

Pop Top of Stack to XT

| SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|------------------------------|---------------------------------|------------------|---------------|
| POP XT | | 1000 0111 1011 1110 | Х | - | 1 |
| Operands XT | Multiplicand register | | | | |
| Description | Predecrement SP by SP -= 2; XT = [SP]; | / 2. Load XT with the 32-bit | value pointec | I to by S | SP: |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this ins | struction follo and executes | ws the only o | e RPT nce. |

PREAD loc16,*XAR7

Read From Program Memory

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|-------------------|---------------------|---------|-----|-----|
| PREAD loc16,*XAR7 | 0010 0100 LLLL LLLL | Х | Y | N+2 |

| Operands | loc16 | Addressing mode (see Chapter 5) |
|--------------------|--|---|
| | *XAR7 | Indirect program-memory addressing using auxiliary register XAR7, can access full 4Mx16 program space range (0x000000 to 0x3FFFFF) |
| Description | | Load the data memory–location pointed to by the "loc16" addressing mode with the 16-bit content of the program–memory location pointed to by "*XAR7": |
| | | [loc16] = Prog[*XAR7]; |
| | | On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*XAR7" addressing mode can be used to access data space variables that fall within the program space address range. |
| | | With some addressing mode combinations, you can get conflicting references. In such cases, the C28x will give the "loc16/loc32" field priority on changes to XAR7. For example: |
| | | PREAD *XAR7,*XAR7 ; *XAR7 given priority PREAD *XAR7++,*XAR7 ; *XAR7++ given priority |
| Flags and Modes | Ν | If (loc16 = $@AX$) and bit 15 of AX is 1, then N is set; otherwise N is cleared. |
| | Z | If (loc16 = $@AX$) and the value of AX is zero, then Z is set; otherwise Z is cleared. |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed N+1 times. When repeated, the "*XAR7" program-memory address is copied to an internal shadow register and the address is post-incremented by 1 during each repetition. |
| Example | ; Copy ; int16 ; ; int16 ; ; for(i ; Array MOVL | <pre>the contents of Array1 to Array2: Array1[N] // Located in program space Array [N] // Located in data space =0; i N; i++) 2[i] = Array1[i]; XAR7,#Array1 ; XAR7 = pointer to Array1</pre> |

| MOVL | XAR2,#Array2 | ; | XAR2 = pointer to Array2 |
|-------|---------------|--------|---------------------------------|
| RPT | #(N-1) | ; | Repeat next instruction N times |
| PREAD | *XAR2++,*XAR7 | ; ; | Array2[i] = Array1[i], i++ |

PUSH ACC

Push Accumulator Onto Stack

| S | YNTAX OI | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|--|--|-------------------------------------|---------------------|---------------|
| PUSH ACC | | | 0001 1110 1011 1101 | X | - | 2 |
| Note: This instru | uction is an a | alieas for the MOV*SP++, A | ACC instruction. | | | |
| Operands | ACC | Accumulator register | r | | | |
| Description | tion Push the 32-bit contents of ACC onto the stack pointed to by SP. Post-increment SP by 2: | | | | | |
| | | [SP] = ACC; SP += 2; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | MOVL XA | R4, #VarA | ; Initialize XAR4 p ; 22-bit address of | oointer with VarA | n the | |
| | MOVL AC | C, *+XAR4[0] | ; Load the 32-bit c ; into ACC | contents of | VarA | |
| | PUSH AC | с | ; Push the 32-bit A ; location pointed ; Post-increment SB | ACC into the to by SP. 2 by 2 | 2 | |

PUSH ARn:ARm

Push 16-bit Auxiliary REgisters Onto Stack

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| PUSH AR1:AR0 | 0111 0110 0000 1101 | Х | - | 1 |
| PUSH AR3:AR2 | 0111 0110 0000 1111 | Х | - | 1 |
| PUSH AR5:AR4 | 0111 0110 0000 1100 | Х | - | 1 |

| Operands | ARn: | AR1:AR0 or AR3:AR2 or AR5:AR4 auxiliary registers |
|----------|------|---|
| | ARm | |

Description Push the contents of two 16-bit auxiliary registers (ARn and ARm) onto the stack pointed to by SP. Post-increment SP by 2:

| PUSH AR1:AR0 [SP] = AR0; [SP+1] = AR1; SP += 2; |
|--|
| PUSH AR3:AR2 [SP] = AR2; [SP+1] = AR3; SP += 2; |
| PUSH AR5:AR4 [SP] = AR4; [SP+1] = AR5; SP += 2; |

Flags and Modes None

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

PUSH AR1H:AR0H

Push AR1H and Ar0H Registers on Stack

| S | SYNTAX OF | TIONS | | OP | CODE | | OBJMODE | RPT | CYC |
|--------------------|------------------------------|--|------------------|----------------------------------|----------------------------------|--------------------------------------|-------------------------------------|---------------------|----------------|
| PUSH AR1H:A | R0H | | 0000 | 0000 | 0000 | 0101 | X | _ | 1 |
| Operands | AR1H: AR0H | Upper 16-bits of XAI | R1 an | d XAR | 0 auxi | iliary reg | jisters | | |
| Description | | Push the contents of pointed to by SP. Post-increment SP to [SP] = AR0H [SP+1] = AR1H; SP += 2; | AROH | H follow | ved by | the con | tents of AR1H | onto th | e stack |
| Flags and Modes | | None | | | | | | | |
| Repeat | | This instruction is instruction, it resets | not ro the re | epeata peat co | ble. I ounter | f this ii r (RPTC | nstruction follo) and executes | ows the s only c | e RPT ince. |
| Example | IntX: | | ; ; | Full inter | conto | ext sav or tra | ve code for ap function | an | |
| | PUSH PUSH PUSH PUSH | AR1H:AR0H XAR2 XAR3 XAR4 | ; ; ; | 16-bi 32-bi 32-bi 32-bi | t AR t sto t sto | 1H and ore of ore of ore of | 16-bit AROH XAR2 XAR3 XAR4 | store | |
| | PUSH PUSH PUSH PUSH | XAR5 XAR6 XAR7 XT | ; ; ; | 32-bi 32-bi 32-bi 32-bi | t sto t sto t sto t sto | ore of ore of ore of ore of | XAR5 XAR6 XAR7 XT | | |
| | | | | | | | | | |

PUSH DBGIER

Push DBGIER Register Onto Stack

| S | YNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|--|--|-----------------|--------------------|----------------|
| PUSH DBGIER | | | 0111 0110 0000 1110 | Х | - | 1 |
| Operands | DBGIER | Debug interrupt en | able register | | | |
| Description | | Push the 16-bit cor Post-increment SP | ntents of DBGIER onto the s by 1: | stack pointed | to by S | SP. |
| | | [SP] = DBGIER; SP += 1; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in s the repeat counter (RPTC | struction follo | ows the es only | e RPT once. |

PUSH DP

Push DP Register Onto Stack

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|---------------------------------|------------------|---------------|
| PUSH DP | | 0111 0110 0000 1011 | Х | - | 1 |
| Operands DP | Data-page register | | | | |
| Description | Push the 16-bit cont Post-increment SP t | ents of DP onto the stack p by 1: | ointed to by S | SP. | |
| | [SP] = DP; SP += 1; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |

PUSH DP:ST1

Push DP and ST1 Onto Stack

| s | YNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|---|--|-----------------|--------------------|----------------|
| PUSH DP:ST1 | | | 0111 0110 0000 1001 | X | - | 1 |
| Operands | DP:ST1 | Data-page register | and status register 1 | | | |
| Description | | Push the 16- bit con the stack pointed to Post-increment SP | ntents of ST1 followed by the o by SP. by 2: | e 16-bit conter | nts of D | P onto |
| | | [SP] = ST1; [SP+1] = DP; SP += 2; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in s the repeat counter (RPTC | struction follo | ows the es only | e RPT once. |

PUSH IFR

Push IFR Onto Stack

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|------------------|---------------------|---------------|
| PUSH IFR | | 0111 0110 0000 1010 | Х | _ | 1 |
| Operands IFR | Interrupt flag registe | r | | | |
| Description | Push the 16-bit cont Post-increment SP t | ents of IFR onto the stack p by 1: | pointed to by \$ | SP. | |
| | [SP] = IFR; SP += 1; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo | ows the s only o | e RPT nce. |

PUSH loc16

Push 16-bit Value on Stack

| S | YNTAX | OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------|--|------------------|--|--|-----------------------|---------------|
| PUSH loc16 | | | 0010 | 0010 LLLL LLLL | Х | _ | 2 |
| | | | | | | | |
| Operands | loc16 | Addressing mode (se | ee Cha | apter 5) | | | |
| Description | | Push a 16-bit value p by SP. Post-increment SP b | ointed y 1: | to by the "loc16" ope | rand on the sta | ack poi | nted to |
| | | [SP] = [loc16]; SP += 1; | | | | | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is instruction, it resets t | not re he rep | peatable. If this inspective to the second s | struction follo and executes | ws the only or | e RPT nce. |
| Example | PUSH (| ΣΩ | ; ; ; | Push the contents the location poin SP. Post-increment | s of XT(31:1 nted to by nt SP by 1 | .5) in | to |
| | PUSH 0 | 9AL | ; ; ; | Push the contents the location poir SP. Post-increment | s of AL onto nted to by nt SP by 1 | o into | |
| | PUSH 0 | DAR4 | ; ; ; | Push the lower 16 the location poin SP. Post-increment | 5-bits of XA nted to by nt SP by 1 | AR4 in | to |
| | PUSH ? | *XAR4++ | ; ; ; | Push the value point othe location by SP. Post-increase by 1 | pinted to by pointed to ement SP and | 7 XAR4) 1 XAR4 | |

PUSH P

Push P Onto Stack

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|---|--|-----------------------------------|---------------------|---------------|
| PUSH P | | | 0111 0110 0001 1101 | Х | - | 1 |
| Operands | Ρ | Product register | | | | |
| Description | | Push the 32-bit cont Post-increment SP b | ents of P onto the stack po by 2: | inted to by SF |) | |
| | | [SP] = P; SP += 2; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | MOVL XA | AR5, #VarA | ; Initialize XAR5 p ; 22-bit address of | pointer with E VarA | n the | |
| | MOVL P, | *+XAR5[0] | ; Load the 32-bit o ; into P | contents of | VarA | |
| | PUSH P | | ; Push the 32-bit 1 ; location pointed ; Post-increment S1 | P into the to by SP. P by 2 | | |

PUSH RPC

Push RPC Onto Stack

| SYNTAX O | PTIONS | OPCODE | OB.IMODE | RPT | CVC |
|--------------------|--|---|-----------------|------------------|---------------|
| STRIANO | | | CDOMODE | 10.1 | 010 |
| PUSH RPC | | 0000 0000 0000 0100 | Х | - | 1 |
| Operands RPC | Return program cou | nter register | | | |
| Description | Push the contents o Post-increment SP b | f the RPC register onto the by 2: | stack pointed | to by S | SP. |
| | [SP] = RPC; SP += 2; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo | ws the only o | e RPT nce. |

PUSH STO

Push ST0 Onto Stack

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|-----------------|---------------------|---------------|
| PUSH ST0 | | 0111 0110 0001 1000 | Х | - | 1 |
| Operands ST0 | Status register 0 | | | | |
| Description | Push the 16-bit cont Post-increment SP t | ents of ST0 onto the stack py 1: | pointed to by | SP. | |
| | [SP] = ST0; SP += 1; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo | ows the s only o | e RPT nce. |

PUSH ST1

Push ST1 Onto Stack

| SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|---------------------|---------|-----|-----|--|
| PUSH ST1 | | 0111 0110 0000 1000 | Х | - | 1 | |
| Operands ST1 | Status register 1 | | | | | |
| Description | Push the 16-bit contents of ST1 onto the stack pointed to by SP. Post-increment SP by 1: | | | | | |
| | [SP] = ST1; SP += 1; | | | | | |
| Flags and Modes | None | | | | | |
| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |

PUSH T:ST0 Push T and ST0 Onto Stack

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|----------------------|------------------|---------------|
| PUSH T:ST0 | | 0111 0110 0001 1001 | Х | - | 1 |
| Operands T:ST0 | The upper 16-bits of | f the multiplicand register ar | nd status regi | ster 0 | |
| Description | Push the 16- bit cont stack pointed to by \$ | ents of ST0 followed by the SP. Post-increment SP by 2 | 16-bit contents : | s of T o | nto the |
| | [SP] = ST0; [SP+1] = T; SP += 2; | | | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this insthe repeat counter (RPTC) | struction follo | ws the only o | e RPT nce. |

PUSH XARn

Push 32-bit Auxiliary Register Onto Stack

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| PUSH XAR0 | 0011 1010 1011 1101 | 1 | - | 1 |
| PUSH XAR1 | 1011 0010 1011 1101 | 1 | - | 1 |
| PUSH XAR2 | 1010 1010 1011 1101 | 1 | - | 1 |
| PUSH XAR3 | 1010 0010 1011 1101 | 1 | - | 1 |
| PUSH XAR4 | 1010 1000 1011 1101 | 1 | - | 1 |
| PUSH XAR5 | 1010 0000 1011 1101 | 1 | - | 1 |
| PUSH XAR6 | 1100 0010 1011 1101 | Х | - | 1 |
| PUSH XAR7 | 1100 0011 1011 1101 | Х | - | 1 |

| Operands | XARn | XAR0 to XAR7, 32-bit auxiliary register |
|----------|------|---|
| | | |

| Description | | Push the 32-bit o Post-increment SP by 2: | contents of XARn onto the stack pointed to by SP. |
|--------------------|--------------------------------------|---|--|
| | | [SP] = XARn; SP += 2; | |
| Flags and Modes | I | None | |
| Repeat | · | This instruction instruction, it res | is not repeatable. If this instruction follows the RPT ets the repeat counter (RPTC) and executes only once. |
| Example | IntX: PUSH PUSH | AR1H:AR0H XAR2 | ; Full context save code for an ; interrupt or trap function ; 16-bit AR1H and 16-bit AR0H store ; 32-bit store of XAR2 |
| | PUSH PUSH PUSH PUSH PUSH | XAR3 XAR4 XAR5 XAR6 XAR7 XT | ; 32-bit store of XAR3 ; 32-bit store of XAR4 ; 32-bit store of XAR5 ; 32-bit store of XAR6 ; 32-bit store of XAR7 ; 32-bit store of XT |
| | • | | |

PUSH XT

Push XT Onto Stack

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|--|--|------------------------------------|---------------------|---------------|
| PUSH XT | | | 1010 1011 1011 1101 | Х | - | 1 |
| Operands | хт | Multiplicand register | | | | |
| Description | | Push the 32-bit c Post-increment SP b | ontents of XT onto the by 2: | stack pointe | d to b | by SP. |
| | | [SP] = XT; SP += 2; | | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example | MOVL XA | R1, #VarA | ; Initialize XAR1 ; ; 22-bit address o | pointer with f VarA | n the | |
| | MOVL XT | , *+XAR5[0] | ; Load the 32-bit | contents of | VarA | |
| | PUSH XT | | ; into XT ; Push the 32-bit X ; location pointed ; Post-increment S | XT into the to by SP. P by 2 | | |

PWRITE *XAR7,loc16

Write to Program Memory

| 5 | SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--|---|---|---|---------------------------------------|-----------------------------|--|
| PWRITE *XAR | 7, loc16 | | 0010 0110 LLLL LLLL | Х | Y | N+5 | |
| Operands | *XAR7 | Indirect program-me can access full 4Mx | ndirect program–memory addressing using auxiliary register XAR7, can access full 4Mx16 program space range (0x000000 to 0x3FFFFF) | | | | |
| | loc16 | Addressing mode (s | ee Chapter 5) | | | | |
| Description | | Load the program- content of the location | memory location pointed to pointed to pointed to by the "loc16" | o by the "*XA addressing n | AR7" w node: | ith the | |
| | | Prog[*XAR7] = [lo | pc16]; | | | | |
| | | On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*XAR7" addressing mode can be used to access data space variables that fall within the program space address range. | | | | | |
| | | With some addres references. In such on changes to XAR7 | sing mode combinations, cases, the C28x will give th 7. For example: | you can g e "loc16/loc32 | et con 2" field | flicting | |
| | | PWRITE *XAR7,* PWRITE *XAR7,*XA | -XAR7 ; *XAR7 c AR7++ ; *XAR7++ c | given priori given priori | lty lty | | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is rep it will be execut program-memory ac address is post-incr | peatable. If the operation foll ted N+1 times. When ddress is copied to an interr remented by 1 during each | ows a RPT in repeated, f nal shadow re repetition. | structio the "* gister <i>a</i> | n, then XAR7" and the | |
| Example | ; Copy ; int16 ; int16 ; for(i ; Arr MOVL MOVL RPT PWRIT | <pre>the contents of A Array1[N]; // L Array2[N]; // L =0; i < N; i++) ay2[i] = Array1[i XAR2,#Array1 XAR7,#Array2 #(N-1) E *XAR7,*XAR2++</pre> | <pre>rray1 to Array2: ocated in data space ocated in program space]; ; XAR2 = pointe: ; XAR7 = pointe: ; Repeat next in ; Array2[i] = A: ; i++</pre> | r to Arrayl r to Array2 nstruction I rray1[i], | N time | S | |

| SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|--------------------|---|--|---|---|--|
| QMACL P,loc32 | 2,*XAR7 | | 0101 0110 0100 1111 1100 0111 LLLL LLLL | 1 | Y | N+2 |
| QMACL P,loc32 | 2,*XAR7++ | | 0101 0110 0100 1111 1000 0111 LLLL LLLL | 1 | Y | N+2 |
| Operands | Р | Product register | | | | |
| | loc32 | Addressing mode (s | ee Chapter 5) | | | |
| | | Note: The @ACC add illegal instruction | lote: The @ACC addressing mode cannot be used when the instruction is repeated. No illegal instruction trap will be generated if used (assembler will flag an error). | | | |
| | *XAR7/ ++ | Indirect program-me can access full 4Mx | ndirect program-memory addressing using auxiliary register XAR7, can access full 4Mx16 program space range (0x000000 to 0x3FFFFF) | | | |
| Description | | 32-bit x 32-bit signed multiply and accumulate. First, add the previous product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. Then, multiply the signed 32-bit content of the location pointed to by the "loc32" addressing mode by the signed 32-bit content of the program-memory location pointed to by the XAR7 register and store the upper 32-bits of the 64-bit result in the P register. If specified, post-increment the XAR7 register by 2: | | | evious ct shift itent of I 32-bit egister ecified, | |
| | | ACC = ACC + P << PM; P = (signed T * signed Prog[*XAR7 or *XAR7++]) >> 32; | | | | |
| | | On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*XAR7/++" addressing mode can be used to access data space variables that fall within the program space address range. | | | | |
| | | With some addressing mode combinations, you can get conflicting references. In such cases, the C28x will give the "loc16/loc32" field priority on changes to XAR7. For example: | | | | |
| | | QMACL P,*XAR [*] QMACL P,*XAR ⁷ +- QMACL P,*XAR ⁷ ,* | 7,*XAR7++ ;XAR7 +,*XAR7 ; *XAR7++ *XAR7++ ; *XAR7++ | given prior given prior given prior | rity rity rity | |
| Flags and Modes | z | After the addition, the | e Z flag is set if the ACC val | ue is zero, els | e Z is c | leared. |
| | N C V OVC | After the addition, the If the addition gener If an overflow occurs If overflow mode is overflow, then the co the operation gene decremented. | e N flag is set if bit 31 of the ates a carry, C is set; other s, V is set; otherwise V is n disabled; and if the oper unter is incremented. If over erates a negative overfl | ACC is 1, else wise C is clea ot affected. ation generat flow mode is c ow, then the | e N is c ared. tes a p lisablec e cour | leared. ositive I; and if hter is |

QMACL P,loc32,*XAR7/++ Signed 32 X 32-bit Multiply and Accumulate (Upper Half)

| OVM | If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7EEEEEE) or maximum negative (0x80000000) if the operation |
|-----|--|
| | overflowed. |
| PM | The value in the PM bits sets the shift mode for the output operation from the |

- product register. If the product shift value is positive (logical left shift operation), then the low bits are zero filled. If the product shift value is negative (arithmetic right shift operation), the upper bits are sign extended.
- **Repeat** This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed N+1 times. The state of the Z, N, C and OVC flags will reflect the final result in the ACC. The V flag will be set if an intermediate overflow occurs in the ACC.

| Example | <pre>; Calcul ; high r ; int32 ; int32 ; int32 ; for(i= ; sum</pre> | ate sum of product us esult: X[N]; // Data info C[N]; // Coefficie: sum = 0; 0; i < N; i++) = sum + ((X[i] * C[i | sin rma nt | g 32-bit multiply and retain ation information (located in low 4M) >> 32) >> 5; |
|---------|---|--|------------------|---|
| | MOVL MOVL SPM ZAPA RPT QMACL | XAR2,#X XAR7,#C -5 #(N-1) P,*XAR2++,*XAR7++ | ;;;;;; | <pre>XAR2 = pointer to X XAR7 = pointer to C Set product shift to ">> 5" Zero ACC, P, OVC Repeat next instruction N times ACC = ACC + P >> 5,</pre> |
| | ADDL MOVL | ACC,P << PM @sum,ACC | ; ; ; ; | <pre>P = (X[1] ^ C[1]) >> 32 i++ Perform final accumulate Store final result into sum</pre> |

| QMPYAL P,X | T,loc32 | 32 Signed 32-bit Multiply (Upper Half) and Add Previous | | | | ous P | |
|--------------------|------------------|--|---|---|--|---|--|
| | | | | | | | |
| | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
| QMPYAL P,XT, | oc32 | | 0101 0110 0100 0110 0000 0000 LLLL LLLL | 1 | - | 1 | |
| Operands | P XT loc32 | Product register Multiplicand register Addressing mode (s | Product register Multiplicand register Addressing mode (see Chapter 5) | | | | |
| Description | | Signed 32-bit x 32-bit multiply and accumulate the previous product. Add the previous signed product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. In addition, multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode and store the upper 32-bits of the 64-bit result in the P register: | | | | | |
| | | ACC = ACC + P << P = (signed T | PM; * signed [loc32]) >> 32 | :; | | | |
| Flags and Modes | z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | | |
| | N C V | After the addition, th If the addition gener If an overflow occur | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. If the addition generates a carry, C is set; otherwise C is cleared. If an overflow occurs, V is set; otherwise V is not affected. | | | | |
| | OVC | If overflow mode is overflow, then the co the operation gen decremented. | If overflow mode is disabled; and if the operation generates a positive overflow, then the counter is incremented. If overflow mode is disabled; and if the operation generates a negative overflow, then the counter is decremented | | | | |
| | OVM | If overflow mode b positive (0x7FFFFF overflowed. | it is set; then the ACC val FF) or maximum negative (0 | ue will satura x80000000) if | ate ma: the ope | ximum eration | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the the product shift value is a low bits are zero filled. If right shift operation), the up | ne output oper positive (log the product oper bits are s | ration fro gical le shift va ign exte | om the ft shift alue is ended. | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this institute the repeat counter (RPTC) | struction follo | ows the s only o | → RPT nce. | |

| Example | ; Calculate signed result: ; Y32 = (X0*C0 + X1*C1 + X2*C2) >> (32 + 2) | | | | |
|---------|---|-------------|---|----------------------------------|--|
| | SPM | -2 | ; | Set product shift mode to ">> 2" | |
| | ZAPA | | ; | Zero ACC, P, OVC | |
| | MOVL | XT,@X0 | ; | XT = X0 | |
| | QMPYL | P,XT,@C0 | ; | P = high 32-bits of (X0*C0) | |
| | MOVL | XT,@X1 | ; | XT = X0 | |
| | QMPYAL | P,XT,@C1 | ; | ACC = ACC + P >> 2, | |
| | | | ; | P = high 32-bits of (X1*C1) | |
| | MOVL | XT,@X2 | ; | XT = X0 | |
| | QMPYAL | P,XT,@C2 | ; | ACC = ACC + P >> 2, | |
| | | | ; | P = high 32-bits of (X2*C2) | |
| | ADDL | ACC,P << PM | ; | ACC = ACC + P >> 2 | |
| | MOVL | @Y32,ACC | ; | Store result into Y32 | |

QMPYL P,XT,loc32

Signed 32 X 32-bit Multiply (Upper Half)

| SYNTAX OPTIONS | | | | OP | CODE | | OBJMODE | RPT | CYC |
|--------------------|------------------|---|------------------|------------------|-------------------|--------------------|---------------------------------|---------------------|----------------|
| QMPYL P,XT,Io | c32 | | 0101 0000 | 0110 0000 | 0110 LLLL | 0111 LLLL | 1 | - | 1 |
| Operands | P XT loc32 | Product register Multiplicand register Addressing mode (see Chapter 5) | | | | | | | |
| Description | | Multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode and store the upper 32-bits of the 64-bit result (a Q30 number) in the P register: | | | | | 32-bit d store er: | | |
| | | P = (signed XT * | sign | ed [lo | bc32]) | >> 32; | | | |
| Flags and Modes | | None | | | | | | | |
| Repeat | | This instruction is instruction, it resets | not re the re | epeata peat c | ble. If ounter | this ins (RPTC) | struction follo and executes | ows the s only a | e RPT ince. |
| Example | ; Calcı | late signed result | t: Y6 | 4 = M | 32*X32 | + B64 | | | |
| - | MOVL | XT,@M32 | ; | XT = | = M32 | | | | |
| | IMPYL | P,XT,@X32 | ; | P = | = low | 32-bits | s of (M32*X3 | 32) | |
| | MOVL | ACC,@B64+2 | ; | ACC = | = high | 32-bit | s of B64 | | |
| | ADDUL | P,@B64+0 | ; | P = | = P + | 10w 32- | -bits of B64 | | |
| | | @164+U,P D XT @X32 | ; | Store | t ⊥OW - hiah | 32-DIT 32_hi+ | result into |) 164 (32) | |
| | ADDCL | ACC.@P | ; | ACC = | - mign = ACC | | arrv | 1941 | |
| | MOVL | @Y64+2,ACC | ; | Store | e high | 32-bit | result int | CO Y64 | |

QMPYL ACC, XT, loc32

Signed 32 X 32-bit Multiply (Upper Half)

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|---|---|---|---------------------------------------|--------------|------------------------------|--|
| QMPYL ACC,X | T,loc32 | | 0101 0110 0110 0011 0000 0000 LLLL LLLL | 1 | - | 2 | |
| Operands | P XT ACC | Product register Multiplicand register Accumulator registe | Product register Multiplicand register Accumulator register | | | | |
| Description | | Multiply the signed 32-bit content of the XT register by the signed 32-bit content of the location pointed to by the "loc32" addressing mode and store the upper 32-bits of the 64-bit result (a Q30 number) in the ACC register: | | | | 32-bit d store gister: | |
| | | ACC = (signed XT) | * signed [loc32]) >> 3 | 32; | | | |
| Flags and Modes | Z | After the operation, the | ne Z flag is set if the ACC va | lue is zero, els | e Z is c | leared. | |
| | Ν | After the operation, the | he N flag is set if bit 31 of the | ACC is 1, els | e N is c | leared. | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; Calcu MOVL IMPYL QMPYL MOVL MOVL | alate signed resul XT,@M32 P,XT,@X32 ACC,XT,@X32 @Y64+0,P @Y64+2,ACC | <pre>t: Y64 = M32*X32 ; XT = M32 ; P = low 32-bit ; ACC = high 32-bit ; Store result into</pre> | cs of (M32*) cs of (M32*) o Y64 | <32) <32) | | |

| QMPYSL P,X | T,loc32 | Signed | d 32-bit Multiply (Upper Hal | f) and Subtra | ct Previ | ous P | | |
|--------------------|------------------|--|--|---|--|--|--|--|
| | | | 000005 | | DDT | 01/0 | | |
| | | | | | RPI | | | |
| | 10032 | | 0000 0000 LLLL LLLL | I | - | 1 | | |
| Operands | P XT loc32 | Product register Multiplicand register Addressing mode (s | Product register Multiplicand register Addressing mode (see Chapter 5) | | | | | |
| Description | | Signed 32-bit x 32-bit multiply and subtract the previous product. Subt the previous signed product (stored in the P register), shifted as specified the product shift mode (PM), from the ACC register. In addition, multiply signed 32-bit content of the XT register by the signed 32-bit constant va and store the upper 32-bits of the 64-bit result in the P register: | | | | ubtract fied by ply the t value | | |
| | | ACC = ACC - P << P = (signed T = | PM; * signed [loc32]) >> 32 | 2; | | | | |
| Flags and Modes | Z | After the subtraction, the Z flag is set if the ACC value is zero, else Z cleared. | | | | se Z is | | |
| | Ν | After the subtractior cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | e N is | | |
| | С | If the subtraction generates a borrow, C is cleared; otherwise C is set. | | | | ət. | | |
| | V | If an overflow occurs | s, V is set; otherwise V is no | ot affected. | | | | |
| | ovc | If overflow mode is overflow, then the co the operation gen decremented. | disabled; and if the oper- punter is incremented. If over erates a negative overflo | ation generat flow mode is c ow, then the | es a p lisablec e cour | ositive l; and if iter is | | |
| | OVM | If overflow mode bi positive (0x7FFFFF overflowed. | it is set; then the ACC val FF) or maximum negative (0 | ue will satura x80000000) if | ate ma the op | ximum eration | | |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the the product shift value is a low bits are zero filled. If right shift operation), the up | ne output oper positive (log the product oper bits are s | ation fr gical le shift va ign exte | om the ft shift alue is ended. | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | PPT nce. | | |

| Example | ; Calculate signed result: ; Y32 = -(X0*C0 + X1*C1 + X2*C2) >> (32 + 2) | | | | | |
|---------|--|-------------|---|----------------------------------|--|--|
| | SPM | -2 | ; | Set product shift mode to ">> 2" | | |
| | ZAPA | | ; | Zero ACC, P, OVC | | |
| | MOVL | XT,@X0 | ; | XT = X0 | | |
| | QMPYL | P,XT,@C0 | ; | P = high 32-bits of (X0*C0) | | |
| | MOVL | XT,@X1 | ; | XT = XO | | |
| | QMPYSL | P,XT,@C1 | ; | ACC = ACC - P >> 2, | | |
| | | | ; | P = high 32-bits of (X1*C1) | | |
| | MOVL | XT,@X2 | ; | XT = X0 | | |
| | QMPYSL | P,XT,@C2 | ; | ACC = ACC - P >> 2, | | |
| | | | ; | P = high 32-bits of (X2*C2) | | |
| | SUBL | ACC,P << PM | ; | ACC = ACC - P >> 2 | | |
| | MOVL | @Y32,ACC | ; | Store result into Y32 | | |

QMPYUL P,XT,loc32

Unsigned 32 X 32-bit Multiply (Upper Half)

| SYNTAX OPTIONS | | | | OP | CODE | | OBJMODE | RPT | CYC |
|--------------------|------------------|--|--|-------------------|-------------------|-------------------------|------------------------------|--------|------------|
| QMPYUL P,XT, | loc32 | | 0101 0000 | 0110 0000 | 0100 LLLL | 0111 LLLL | 1 | - | 1 |
| Operands | P XT loc32 | Product register Multiplicand register Addressing mode (se | Product register Aultiplicand register Addressing mode (see Chapter 5) | | | | | | |
| Description | | Multiply the unsigned 32-bit content of the XT register by the unsigned 32-bit content of the location pointed to by the "loc32" addressing mode and store the upper 32-bits of the 64-bit result in the P register: | | | | ed 32-bit store the | | | |
| | | P = (unsigned XT) | * un | signe | l [loc | :32]) >> | 32; | | |
| Flags and Modes | | None | | | | | | | |
| Repeat | | This instruction is no it resets the repeat c | t repe ounte | atable er (RP1 | If this C) and | instructio d execute | on follows the es only once. | RPT in | struction, |
| Example | ; Calcı | late unsigned resu | ult: | Y64 = | M32*X | K32 + B6 | 54 | | |
| | MOVL | XT,@M32 | ; | XT : | = M32 | | | | |
| | IMPYL | P,XT,@X32 | ; | P = | = low | 32-bits | of (M32*X3 | 32) | |
| | MOVL | ACC,@B64+2 | ; | ACC : | = higł | n 32-bit | s of B64 | | |
| | ADDUL | P,@B64+0 | ; | P : | = P + | low 32- | bits of B64 | Ł | |
| | MOVL | @Y64+0,P | ; | Store | e low | 32-bit | result into | 9 Y64 | |
| | QMPYUL | P,XT,@X32 | ; | P : | = higł | n 32-bit | s of (M32*X | (32) | |
| | ADDCL | ACC,@P | ; | ACC : | = ACC | + P + c | arry | | |
| | MOVL | @Y64+2,ACC | ; | Store | e higł | ı 32-bit | result int | o Y64 | |

QMPYXUL P,XT,loc32

Signed X Unsigned 32-bit Multiply (Upper Half)

| SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|---|---|---|---|--------------------------------------|---------------------|
| QMPYXUL P,XT,loc32 | | 0101 0110 0100 0010 | 1 | - | 1 |
| | | 0000 0000 LLLL LLLL | | | |
| Operands P XT loc32 | Product register Multiplicand register Addressing mode (s | see Chapter 5) | | | |
| Description | Multiply the signed 3 content of the location the upper 32-bits of | 32-bit content of the XT region pointed to by the "loc32" of the 64-bit result in the P re | ster by the ur addressing m gister: | nsignec ode an | l 32-bit d store |
| | P = (signed XT * | unsigned [loc32]) >> 3 | 32; | | |
| Flags and Modes | None | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. |
| Example ; Calcu ; Y64 = MOVL QMPYXUI MOV LSL64 ASR64 MOVL MOVL QMPYXUI MOV LSL64 ASR64 MOVL USL64 ASR64 MOVL MOVL MOVL MOVL MOVL ADDUL ADDUL ADDUL ADDUL ADDUL ADDUL ADDUL | alate signed resul Y1:Y0, M64 = M1: XT,@X1 ; X P,XT,@M0 ; F @T,#32 ; T ACC:P,T ; A @XAR4,P ; X @XAR5,ACC ; X XT,@M1 ; X @XAR5,ACC ; A XT,@M1 ; X @XAR5,ACC ; F @XAR6,P ; X @XAR6,P ; X @XAR7,ACC ; F ACC,@XAR5 ; F ACC,@XAR5 ; F ACC,@AR6 ; F ACC,@AR7 ; F P,@B0 ; F ACC,@B1 ; S | t: Y64 = (M64*X64) >> 6 M0, X64 = X1:X0, B64 = XT = X1 P = high 32-bits of (T = 32 ACC:P = ACC:P << T ACC:P = ACC:P >> T XAR5:XAR4 = ACC:P XT = M1 P = high 32-bits of (T = 32 ACC:P = ACC:P << T ACC:P = ACC:P >> T XAR7:XAR6 = ACC:P P = low 32-bits of (ACC = high 32-bits of (ACC:P = ACC:P + XAR7:XA ACC:P = ACC:P + B64 Store result into Y64 | 54 + B64 B1:B0 uns M0 * si sign M1 * u sign M1 * s sign M1 * s R4 R6 | gn X1) ns X0) ign X1 ign X1 | _) _) |

ROL ACC

Rotate Accumulator Left

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ROLACC | 1111 1111 0101 0011 | Х | Y | N+1 |

Operands ACC Accumulator register

Rotate the content of the ACC register left by one bit, filling bit 0 with the content of the carry flag and loading the carry flag with the bit shifted out: Description



| Flags and Modes | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
|--------------------|---|---|
| | Z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. |
| | С | The value in bit 31 of the ACC register is transferred to C. The value in C before the rotation is transferred to bit 0 of the ACC. |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then |

| - | the ROL instruction will be executed N+1 times. The state of the Z, N, and C |
|---------|--|
| | flags will reflect the final result. |
| Evennle | Detete contents of New Jeft by C |

| ; Rolale | contents of | VarA lelt by 5: |
|----------|--|---|
| MOVL | ACC,@VarA | ; ACC = VarA |
| RPT | #4 | ; Repeat next instruction 5 times |
| ROL | ACC | ; Rotate ACC left |
| MOVL | @VarA,ACC | ; Store result into VarA |
| | ; ROCALE MOVL RPT ROL MOVL | , Rotate contents of MOVL ACC,@VarA RPT #4 ROL ACC MOVL @VarA,ACC |

ROR ACC

Rotate Accumulator Right

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| ROR ACC | 1111 1111 0101 0010 | Х | Y | N+1 |

Operands ACC Accumulator register

Description Rotate the content of the ACC register right by one bit, filling bit 31 with the content of the carry flag and loading the carry flag with the bit shifted out:



| Flags and Modes | Ν | After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. |
|--------------------|---|---|
| | z | After the operation, the Z flag is set if the ACC is zero, else Z is cleared. |
| | С | The value in bit 0 of the ACC register is transferred to C. The value in C before the rotation is transferred to bit 31 of the ACC. |

RepeatThis instruction is repeatable. If the operation follows a RPT instruction, then
the ROR instruction will be executed N+1 times. The state of the Z, N, and C
flags will reflect the final result.

| Example | ; | Rotate | contents of | VarA | right | by 5: |
|---------|---|--------|-------------|------|-------|---------------------------------|
| | | MOVL | ACC,@VarA | | ; | ACC = VarA |
| | | RPT | #4 | | ; | Repeat next instruction 5 times |
| | | ROR | ACC | | ; | Rotate ACC right |
| | | MOVL | @VarA,ACC | | ; | Store result into VarA |
RPT #8bit/loc16

Repeat Next Instruction

| SYNTAX OPTIONS | | | | OPCODE | | OBJMODE | RPT | CYC | |
|--------------------|--|---|--|---|---|---|-----------|---|--|
| RPT #8bit | | | 1111 0 | 110 CCCC | CCCC | Х | - | 1 | |
| RPT loc16 | | | 1111 0 | 111 LLLL | LLLL | Х | - | 4 | |
| Operands | #8bit loc16 | 8-bit constant immedia Addressing mode (see | B-bit constant immediate value (0 to 255 range) Addressing mode (see Chapter 5) | | | | | | |
| Description | | Repeat the next instruction. An internal repeat counter (RPTC) is loaded with a value N that is either the specified #8bit constant value or the content of the location pointed to by the "loc16" addressing mode. After the instruction that follows the RPT is executed once, it is repeated N times; that is, the instruction following the RPT executes $N + 1$ times. Because the RPTC cannot be saved during a context switch, repeat loops are regarded as multicycle instructions and are not interruptible. | | | | | | a value pointed is exe- xecutes eat loops | |
| | | Note on syntax: | | | | | | | |
| | | Parallel bars () before the instruction is rep | ore the re beated ar | epeated ins nd is not ir | struction iterruptat | are used as a ble. | remino | ler that | |
| | | When writing inline a | assembly | y, use the | syntax | | | | |
| | | <pre>asm(RPT #8bt/ loc16 instruction");</pre> | | | | | | | |
| | | Not all instructions are repeatable. If an instruction that is not repeatable follows the RPT instruction, the RPTC counter is reset to 0 and the instruction only executes once. The 28x Assembly Language tools check for this condition and issue warnings. | | | | | | ows the kecutes arnings. | |
| Flags and Modes | | None | | | | | | | |
| Repeat | | This instruction is not r sets the repeat counte | repeatable er (RPTC) | e. If this inst and execu | ruction fol tes only o | lows the RPT in nce. | nstructio | on, it re- | |
| Example | ; Copy ; to ; ; int ; int ; for ; Arr, MOVI RPT | y the number of el Array2: 16 Array1[N]; // Lo 16 Array2[N]; // Lo (i=0; i < VarA; i+ ay2[i] = Array1[i] L XAR2,#Array2 @VarA EAD *XAR2++,*(Array | <pre>ements ocated ocated ++) ; y1)</pre> | <pre>specifie in high in data ; XAR2 = ; Repeat ; [VarA] ; Array2</pre> | d in Va 64K of p space pointer next in + 1 tir [i] = An | rA from Arr program space r to Array2 nstruction nes rray1[i], | ayl ce | | |

SAT ACC

Saturate Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| SAT ACC | 1111 1111 0101 0111 | Х | - | 1 |
| | | | | |

| Operands | ACC | Accumulator register | |
|--------------------|---------|---|---|
| Description | | Saturate the ACC register overflow counter (OVC): | to reflect the net overflow represented in the 6-bit |
| | | <pre>if(OVC > 0) ACC = 0x7FFF FFFF; V = 1; if(OVC < 0) ACC = 0x8000 0000; V = 1; if(OVC = 0) ACC = unchanged; OVC = 0;</pre> | |
| Flags and Modes | Ν | After the operation, the N fla | ag is set if bit 31 of the ACC is 1, else N is cleared. |
| | Z | After the operation, the Z f | lag is set if the ACC is zero, else Z is cleared. |
| | С | C is cleared. | |
| | v | If (OVC != 0) at the start of | f the operation, V is set; otherwise, V is cleared |
| | OVC | If $(OVC > 0)$ then ACC is a If $(OVC < 0)$ then ACC is a if $(OVC = 0)$ then ACC is a After the operation, OVC is | eaturated to its maximum positive value. saturated to its maximum negative value. not modified. s cleared. |
| Repeat | | This instruction is not re instruction, it resets the re | epeatable. If this instruction follows the RPT peat counter (RPTC) and executes only once. |
| Example | ; Add V | arA, VarB and VarC and | saturate result and store in VarD: |
| | ZAP | OVC | ; Clear overflow counter |
| | MOVL | ACC,@VarA | ; Load ACC with contents of VarA |
| | ADDL | ACC,@VarB | ; Add to ACC contents of VarB |
| | ADDL | ACC,@VarC | ; Add to ACC contents of VarC |
| | SAT | ACC | ; Saturate ACC based on OVC value |
| | MOVL | @VarD,ACC | ; Store result into VarD |

| SAT64 ACC:P Saturate 64-bit Value ACC: | | | | | | | | |
|--|----------|--|--|---------------------------------|------------------|------------|--|--|
| 5 | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
| SAT64 ACC:P | | | 0101 0110 0011 1110 | 1 | - | 1 | | |
| Operands | ACC:P | Accumulator registe | r (ACC) and product registe | er (P) | | | | |
| Description | | Saturate the 64-bit co overflow represented | ontent of the combined ACC d in the overflow counter (C | :P registers to VC): | reflect | the net | | |
| | | <pre>if(OVC > 0) ACC:P = 0x7FFF V=1; if(OVC < 0) ACC:P = 0x8000 V=1; if(OVC = 0) ACC:P = unchag OVC = 0;</pre> | F FFFF FFFF FFFF; 0 0000 0000 0000; ged; | | | | | |
| Flags and Modes | Ν | After the shift, if bit 3 and the N bit is set; | After the shift, if bit 31 of the ACC register is 1 then ACC:P is negative and the N bit is set: otherwise N is cleared. | | | | | |
| | Z | After the operation, t is zero; otherwise, Z | he Z flag is set if the combine | ed 64-bit value | e of the | ACC:P | | |
| | С | The C bit is cleared. | | | | | | |
| | v | At the start of the op is set. | peration, if $(OVC = 0)$ then $\$ | / is cleared; o | therwis | se, V | | |
| | OVC | If (OVC = 0), then no ACC:P is uncha | saturation takes place: anged. | | | | | |
| | | If(OVC > 0), then sat ACC:P = 0x7Ff | urate ACC:P the maximum FF FFFF FFFF FFFF | positive value | e: | | | |
| | | If(OVC < 0), then sa ACC = 0x8000 | turate ACC:P to the maxim 0000 or ACC:P = 0x8000 0 | um negative 0000 0000 000 | value: 00 | | | |
| | | At the end of the ope | eration, OVC is cleared. | | | | | |
| Repeat | | This instruction is no struction, it resets th | ot repeatable. If this instruct le repeat counter (RPTC) a | ion follows th nd executes o | e RPT only on | in- ce. | | |

| Example | ; Add | 64-bit VarA, Va | arB a | nd VarC, sat and store result in VarD: |
|---------|-------|-----------------|-------------|---|
| | ZAP | OVC | ; | Clear overflow counter |
| | MOVL | P,@VarA+0 | ; | Load P with low 32-bits of VarA |
| | ADDUL | P,@VarB+0 | ; | Add to P unsigned low 32-bits of VarB |
| | ADDUL | P,@VarC+0 | ; | Add to P unsigned low 32-bits of VarC |
| | MOVU | @AL,OVC | ; ; ; | Store overlow (repeated carry) in the ACC and then add higher portion of the 64 bit variables |
| | MOVB | AH,#0 | ; ; ; | Store overlow (repeated carry) in the ACC and then add higher portion of the 64 bit variables |
| | ZAP | OVC | ; | Clear overflow counter |
| | ADDL | ACC,@VarA+2 | ; | Add to ACC with carry high 32-bits of VarA |
| | ADDL | ACC,@VarB+2 | ; | Add to ACC with carry high 32-bits of VarB |
| | ADDL | ACC,@VarC+2 | ; | Add to ACC with carry high 32-bits of VarC |
| | SAT64 | ACC:P | ; | Saturate ACC:P based on OVC value |
| | MOVL | @VarD+0,P | ; | Store low 32-bit result into VarD |
| | MOVL | @VarD+2,ACC | ; | Store high 32-bit result into VarD |

SB 8bitOffset,COND

| S | Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο | ONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------------------------------|---|---|---|--|--------------------|------------------|
| SB 8bitOffset,COND | | | | 0110 COND CCCC CCCC | Х | - | 7/4 |
| Operands | 8bitOffset | 8-bit s +127 ra | 8-bit signed immediate constant offset value (-128 t +127 range) | | | | |
| | COND | Condit | ional co | odes: | | | |
| | | COND | Syntax | Description | Flags | Teste | b |
| | | 0000 | NEQ | Not Equal To | Z = 0 | | |
| | | 0001 | EQ | Equal To | Z = 1 | | |
| | | 0010 | GT | Greater Then | Z = 0 | AND N | = 0 |
| | | 0011 | GEQ | Greater Then Or Equal | To $N = 0$ | | |
| | | 0100 | LT | Less Then | N = 1 | | |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 | OR N : | = 1 |
| | | 0110 | HI | Higher | C = 1 | AND Z | = 0 |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | |
| | | 1001 | LOS | Lower Or Same | C = 0 | OR Z : | = 1 |
| | | 1010 | NOV | No Overflow | V = 0 | | |
| | | 1011 | OV | Overflow | V = 1 | | |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 |) | |
| | | 1101 | TC | Test Bit Set | TC = 1 | - | |
| | | 1110 | NBIO | BIO Input Equal To Ze | ro BIO = | 0 | |
| | | 1111 | UNC | Unconditional | - | | |
| Description | | Short co by addi otherwis | onditional ng the s se contin | l branch. If the specified col igned 8-bit constant value ue execution without brancl | ndition is true to the curre hing: | , then l ent PC | oranch value; |
| | | If (CON If (CON | ND = tru ND = fal | ne) PC = PC + signed 8- .se) PC = PC + 1; | bit offset | ; | |
| | | Note: If (COND = true) then the instruction takes 7 cycles. If (COND = false) then the instruction takes 4 cycles. If (COND = UNC) then the instruction takes 4 cycles. | | | | | |
| Flags and Modes | v | If the V | flag is te | sted by the condition, then ' | V is cleared. | | |
| Repeat | | This ins instructi | struction on, it rese | is not repeatable. If this in ets the repeat counter (RPT) | struction foll C) and execu | ows th tes only | e RPT / once. |

SBBU ACC, loc16

Subtract Unsigned Value Plus Inverse Borrow

| | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|--|---|--|---------------------|------------------|--|--|
| SBBU ACC, loc | :16 | | 0001 1101 LLLL LLLL | Х | - | 1 | | |
| Operands | ACC loc16 | Accumulator registe Addressing mode (s | er See Chapter 5) | | | | | |
| Description | | Subtract the 16-bit addressing mode, z flag bit from the AC | Subtract the 16-bit contents of the location pointed to by the "loc16 addressing mode, zero extended, and subtract the compliment of the carry flag bit from the ACC register: | | | | | |
| | | ACC = ACC - 0: [1] | oc16] - ~C; | | | | | |
| Flags and Modes | Z | After the subtractior | n, the Z flag is set if ACC is | zero, else Z is | s cleare | ∍d. | | |
| | Ν | After the subtraction cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | se N is | | |
| | С | The state of the carry subtraction generate | y bit before execution is inclues a borrow, C is cleared; o | ided in the sub therwise C is | otractior set. | 1. If the | | |
| | V | If an overflow occur | s, V is set; otherwise V is n | ot affected. | | | | |
| | ovc | If(OVM = 0, disable then the counter is i overflow, then the c | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the | | | | | |
| | ΟνΜ | lf overflow mode b positive (0x7FFFFI overflowed. | it is set; then the ACC va FF) or maximum negative (0 | lue will satur 0x80000000) if | ate ma the op | ximum eration | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT | | |
| Example | ; Sub MOVU ADD SUBU SUB SBBU SUB | tract three 32-bit ACC,@VarAlow ACC,@VarAhigh << 1 ACC,@VarBlow ACC,@VarBhigh << 1 ACC,@VarClow ACC,@VarClow | unsigned variables by ; AH = 0, AL = Var 6 ; AH = VarAhigh, A ; ACC = ACC - 0:Va 6 ; ACC = ACC - VarB ; ACC = ACC - VarC 6 ; ACC = ACC - VarC | 16-bit part CAlow AL = VarAlow ArBlow Bhigh << 16 Clow - ~Carn Chigh << 16 | s: / | | | |
| | | | , | | | | | |

SBF 8bitOffset,EQ/NEQ/TC/NTC

Short Branch Fast

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------------|---------|-----|-----|
| SBF 8bitOffset,EQ | 1110 1100 CCCC CCCC | 1 | I | 4/4 |
| SBF 8bitOffset,NEQ | 1110 1101 CCCC CCCC | 1 | - | 4/4 |
| SBF 8bitOffset,TC | 1110 1110 CCCC CCCC | 1 | - | 4/4 |
| SBF 8bitOffset,NTC | 1110 1111 CCCC CCCC | 1 | - | 4/4 |

| Operands | 8bitOffset | 8-bit signed immediate constant offset value (-128 to +127 range) | | | | |
|--------------------|------------|---|--|--|--|--|
| | | Syntax | Description | Flags Tested | | |
| | | NEQ | Not Equal To | Z = 0 | | |
| | | EQ | Equal To | Z = 1 | | |
| | | NTC | Test Bit Not Set | TC = 0 | | |
| | | TC | Test Bit Set | TC = 1 | | |
| Description | | Short fa branch b otherwis | <pre>Short fast conditional branch. If the specified condition is true, then branch by adding the signed 8-bit constant value to the current PC value; otherwise continue execution without branching: If (tested condition = true) PC = PC + signed 8-bit off- set; If (tested condition = false) PC = PC + 1; Note: The short branch fast (SBF) instruction takes advantage of dual pre-fetch queue on the C28x core that reduces the cycles for a taken branch from 7 to 4:</pre> | | | |
| | | If (tes set; If (tes | | | | |
| | | Note: T o | | | | |
| | | lf If | (tested condition = true) then the instru- (tested condition = false) then the instru- | uction takes 4 cycles. ruction takes 4 cycles. | | |
| Flags and Modes | | None | | | | |
| Repeat | | This ins | truction is not repeatable. If t | his instruction follows the RPT (RPTC) and executes only once. | | |

SBRK #8bit

Subtract From Current Auxiliary Register

| | SYNTAX OF | TIONS | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--------------|--|---|---|---------------------------------|---------------------|-------|--|--|
| SBRK,#8bit | | | 1111 | 1101 CCCC CCCC | Х | - | 1 | | |
| Operands | #8bit | 8-bit constant imme | diate | value | | | | | |
| Description | | Subtract the 8-bit ur ARP: | Subtract the 8-bit unsigned constant from the XARn register pointed to by ARP: | | | | | | |
| | | XAR(ARP) = XAR(AR) | RP) — | 0:8bit; | | | | | |
| Flags and Modes | ARP | The 3-bit ARP point This pointer determin | The 3-bit ARP points to the current valid auxiliary register, XAR0 to XAR7. This pointer determines which auxiliary register is modified by the operation. | | | | | | |
| Repeat | | This instruction is instruction, it resets | not r the re | epeatable. If this in epeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT | | |
| Example | | .word 0xEEEE | | | | | | | |
| | | .word 0x0000 | | | | | | | |
| | TableA: | .word 0x1111 | | | | | | | |
| | | .word 0x2222 | | | | | | | |
| | | .word 0x3333 | | | | | | | |
| | | .word 0x4444 | | | | | | | |
| | FuncA: | | | | | | | | |
| | MOVI | L XAR1,#TableA | ; | Initialize XAR1 p | pointer | | | | |
| | MOV | Z AR2,*XAR1 | ; ; | Load AR2 with the pointed to by XAR | e 16-bit va R1 (0x1111) | lue | | | |
| | | | ; | Set ARP = 1 | | | | | |
| | | | ; | Decrement XAR1 by | 7 2 | | | | |
| | SBRK MOVZ | #2 AR3,*XAR1 | ; | Load AR3 with the pointed to by XAB | e 16-bit val R1 (0xEEEE) | Lue | | | |

SETC Mode

Set Multiple Status Bits

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| SETC Mode | 0011 1011 CCCC CCCC | Х | - | 1,2 |
| SETC SXM | 0011 1011 0000 0001 | Х | - | 1 |
| SETC OVM | 0011 1011 0000 0010 | Х | - | 1 |
| SETC TC | 0011 1011 0000 0100 | Х | - | 1 |
| SETC C | 0011 1011 0000 1000 | Х | - | 1 |
| SETC INTM | 0011 1011 0001 0000 | Х | - | 2 |
| SETC DBGM | 0011 1011 0010 0000 | Х | - | 2 |
| SETC PAGE0 | 0011 1011 0100 0000 | Х | - | 1 |
| SETC VMAP | 0011 1011 1000 0000 | Х | - | 1 |

Operands mode 8-bit immediate mask (0x00 to 0xFF)

Description

Set the specified status bits. The "mode" operand is a mask value that relates to the status bits in this way:

| "Mode" bit | Status Register | Flag | Cycles |
|------------|-----------------|-------|--------|
| 0 | ST0 | SXM | 1 |
| 1 | ST0 | OVM | 1 |
| 2 | ST0 | TC | 1 |
| 3 | ST0 | С | 1 |
| 4 | ST1 | INTM | 2 |
| 5 | ST1 | DBGM | 2 |
| 6 | ST1 | PAGE0 | 1 |
| 7 | ST1 | VMAP | 1 |

 Note:
 The assembler will accept any number of flag names in any order. For example:

 SETC
 INTM,TC
 ; Set INTM and TC bits to 1

 SETC
 TC,INTM,OVM,C
 ; Set TC, INTM, OVM, C bits to 1

 Flags and
Modes
 SXM
 Any of the specified bits can be set by the instruction.

 OVM
TC
C
INTM
DBGM
PAGE0
VMAP
 Any of the specified bits can be set by the instruction.

Repeat

This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once

| Example | ; Mod | ify flag settings: | | |
|---------|-------|--------------------|---|---------------------------------|
| | SETC | INTM, DBGM | ; | Set INTM and DBGM bits to 1 |
| | CLRC | TC,C,SXM,OVM | ; | Clear TC, C, SXM, OVM bits to 0 |
| | CLRC | #0xFF | ; | Clear all bits to 0 |
| | SETC | #0xFF | ; | Set all bits to 1 |
| | SETC | C,SXM,TC,OVM | ; | Set TC, C, SXM, OVM bits to 1 |
| | CLRC | DBGM, INTM | ; | Clear INTM and DBGM bits to 0 |
| | | | | |

SETC MOM1MAP

Set the M0M1MAP Status Bit

| S | YNTAX OPT | IONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|--|--|-----------------------------------|--------------------|------------------|
| SETC MOM1MA | ٩P | | 0101 0110 0001 1010 | Х | - | 5 |
| Operands | MOM1MAP | Status bit | | | | |
| Description | | Set the M0M1MA memory blocks f mapped as follow | P status bit, configuring the for C28x/C2XLP operation /s: | mapping of th . The memor | ne M0 a ry bloc | and M1 ks are |
| MOM1MAP | bit | Data Space | F | Program Space | e | |
| 0 | | M0: 0x000 to 0x | 3FF MC |): 0x400 to 0x7 | FF | |
| (C27x) | | M1: 0x400 to 0x | 7FF M1 | : 0x000 to 0x3 | FF | |
| 1 | | | M0: 0x000 to 0x3FF | | | |
| (C28x/C2XL | _P) | | M1: 0x400 to 0x7FF | | | |
| Note: The pipe | line is flushed v | when this instruction is e | executed. | | | |
| Flags and Modes | MOM1MAP | The M0M1MAP b | it is set. | | | |
| Repeat | | This instruction i instruction, it rese | s not repeatable. If this ir ets the repeat counter (RPT) | nstruction follo C) and execut | ows th tes only | e RPT / once. |
| Example | ; Set th Reset: SETC CLRC .c28_a SETC | e device mode OBJMODE ; E AMODE ; E mode ; I MOM1MAP ; E | from reset to C28x: Enable C28x Object Mode Enable C28x Address Mod Cell assembler we are i Enable C28x Mapping Of | e n C28x addr M0 and M1 b | ess mo locks | ode |
| | | | | | | |

SETC OBJMODE

Set the OBJMODE Status Bit

| S | SYNTAX OPTI | ONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--|--|--|----------------------------------|--------------------|------------------|--|--|
| SETC OBJMO | DE | | 0101 0110 0001 1111 | Х | - | 5 | | |
| Operands | OBJMODE | Status bit | | | | | | |
| Description | | Set the OBJMODE status bit, putting the device in C28x object mod (supports C2XLP source): | | | | | | |
| Flags and Modes | OBJMODE | Set the OBJMOD | Set the OBJMODE bit. | | | | | |
| Repeat | | This instruction i instruction, it rese | s not repeatable. If this in ets the repeat counter (RPT) | struction follo C) and execut | ows th tes only | e RPT / once. | | |
| Example | ; Set the Reset: SETC (CLRC 2 .c28_an SETC 1 | device mode fr DBJMODE ; E AMODE ; E node ; T MOM1MAP ; E | om reset to C28x: Enable C28x Object Mode Enable C28x Address Mod Cell assembler we are i Enable C28x Mapping Of | e n C28x addr M0 and M1 b | ess mo locks | ode | | |

SETC XF

Set XF Bit and Output Signal

| 5 | SYNTAX OP | TIONS | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---|--|----------------------|--|--|-----|-----|--|--|
| SETC XF | | | 0101 | 0110 0010 0110 | Х | - | 1 | | |
| Operands | XF | Status bit and output signal | | | | | | | |
| Description | \$ | Set the XF status bit and pull the corresponding output signal high. | | | | | | | |
| Flags and Modes | XF | The XF status bit is set. | | | | | | | |
| Repeat | - i | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Pulse : MOV SB SET CLR Dest: | KF signal high i AL,@VarA Dest,NEQ C XF C XF | f bra ; ; ; | nch not taken: Load AL with cont ACC = VarA Set XF bit and s: Clear XF bit and | cents of Van ignal high signal low | сA | | | |
| | DCBC. | | | | | | | | |

SFR ACC,#1..16

Shift Accumulator Right

| SYNTAX OPTIONS | | | OF | CODE | OBJMODE | RPT | CYC | |
|--------------------|--|---|--|--|--|-------------|------------------|--|
| SFR ACC,#11 | 16 | | 1111 1111 | 0100 SHFT | Х | Y | N+1 | |
| Operands | ACC #116 | Accumulator registe Shift value | r | | | | | |
| Description | | Right shift the conter field. The type of shi sign extension mode | Right shift the content of the ACC register by the amount specified in the shift field. The type of shift (arithmetic or logical) is determined by the state of the sign extension mode (SXM) bit: | | | | | |
| | | <pre>if(SXM = 1)</pre> | | | | | led bled | |
| Flags and Modes | Z | After the shift, the Z | flag is set if | the ACC value | is zero, else | Z is cle | ared. | |
| | Ν | After the shift, the N | flag is set i | f bit 31 of the A | CC is 1, else l | N is cle | ared. | |
| | С | The last bit shifted o | out is loaded | l into the C flag | bit. | | | |
| | SXM | If $(SXM = 1)$, then the operation behaves like an arithmetic right shift. If $(SXM = 0)$, then the operation behaves like a logical right shift. | | | | | ft. | |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, ther the SFR instruction will be executed $N+1$ times. The state of the Z, N and C flags will reflect the final result. | | | | | n, then and C | |
| Example | ; Arith MOVL SETC SFR MOVL | nmetic shift right ACC,@VarA SXM ACC,#10 @VarA,ACC | contents ; AC ; Er ; Ar ; St | of VarA by 1 CC = VarA nable sign ex rithmetic shi core result i | 0: tension mode ft right ACe nto VarA | e C by 1 | .0 | |

SFR ACC,T

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|---------|--|---|---|----------------------------------|---------------------------------|--|--|
| SFR ACC,T | | | 1111 1111 0101 0001 | Х | - | 1 | | |
| • | | | | | | | | |
| Operands | ACC | Accumulator registe | r | | | | | |
| | т | Upper 16-bits of the | multiplicand (XT) register | | | | | |
| Description | | Right shift the conter least significant bits ignored. The type of the sign extension n | nt of the ACC register by the of the T register, T(3:0) = (shift (arithmetic or logical) is node (SXM) bit: | amount speci 015. Higher s determined l | fied in t order b by the s | he four bits are state of | | |
| | | <pre>if(SXM = 1) // sign extension mode enabled ACC = S:ACC >> T(3:0); // arithmetic shift right else // sign extension mode disabled ACC = 0:ACC >> T(3:0); // logical shift right</pre> | | | | | | |
| Flags and Modes | Z | After the shift, the Z flag is set if the ACC value is zero, else Z is cleared. Even if the T register specifies a shift of 0, the content of the ACC register is still tested for the zero condition and Z is affected. | | | | | | |
| | Ν | After the shift, the N t if the T register spec tested for the negati | flag is set if bit 31 of the ACC cifies a shift of 0, the conten ve condition and N is affect | is 1, else N is It of the ACC ed. | cleared registe | l. Even r is still | | |
| | С | If $(T(3:0) = 0)$ then C into the C flag bit. | is cleared; otherwise, the | last bit shifted | out is | loaded | | |
| | SXM | if (SXM = 1), then th If (SXM = 0), then th | e operation behaves like an le operation behaves like a | n arithmetic ri logical right s | ght shif hift. | ť. | | |
| Repeat | | This instruction is repeatable. If the operation follows a RPT instruction, then the SFR instruction will be executed N+1 times. The state of the Z, N and C flags will reflect the final result. | | | | | | |
| Example | ; Arith | metic shift right | contents of VarA by VarA | arB: | | | | |
| | , MOVL | ACC,@VarA | : ACC = VarA | | | | | |
| | MOV | T,@VarB | ; $T = VarB$ (shift | t value) | | | | |
| | SETC | SXM | ; Enable sign ext | tension mode | e | | | |
| | SFR | ACC,T | ; Arithmetic shi: | Et right AC | C by T | (3:0) | | |
| | MOVL | @VarA,ACC | ; Store result in | nto VarA | - | | | |

SPM shift

Set Product Mode Shift Bits

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--|---------------------|---------|-----|-----|
| SPM +1 | 1111 1111 0110 1000 | Х | - | 1 |
| SPM 0 | 1111 1111 0110 1001 | Х | - | 1 |
| SPM –1 | 1111 1111 0110 1010 | Х | - | 1 |
| SPM –2 | 1111 1111 0110 1011 | Х | - | 1 |
| SPM –3 | 1111 1111 0110 1100 | Х | - | 1 |
| SPM -4 (Valid only when AMODE = 0) SPM +4 (Valid only when AMODE = = 1) | 1111 1111 0110 1101 | Х | - | 1 |
| SPM –5 | 1111 1111 0110 1110 | Х | - | 1 |
| SPM –6 | 1111 1111 0110 1111 | X | - | 1 |

Operands shift Product shift mode (+4, +1, 0, -1, -2, -3, -4, -5, -6)

Description

Specify a product shift mode. A negative value indicates an arithmetic right shift; positive numbers indicate a logical left shift. The following table shows the relationship between the "shift" operand and the 3-bit value that gets loaded into the product shift mode (PM) bits in ST0. The address mode bit (AMODE) selects between two types of shift decodes as shown in the table below:

| PM Bits | AMODE = 1 | $\mathbf{AMODE} = 0$ |
|---------|-----------|----------------------|
| 000 | SPM +1 | SPM +1 |
| 001 | SPM 0 | SPM 0 |
| 010 | SPM –1 | SPM -1 |
| 011 | SPM –2 | SPM –2 |
| 100 | SPM –3 | SPM –3 |
| 101 | SPM +4 | SPM -4 |
| 110 | SPM –5 | SPM –5 |
| 111 | SPM -6 | SPM –6 |

Flags and Modes ΡM

Repeat

This instruction is repeatable. If the operation follows a RPT instruction, then the SFR instruction will be executed N+1 times. The state of the Z, N and C flags will reflect the final result.

PM is loaded with the 3-bit value specified by the selected "shift" value.

Example ; Calculate: Y32 = M16*X16 >> 4 + B32 CLRC AMODE ; Make sure AMODE = 0 SPM $^{-4}$; Set product shift mode to ">> 4" T,@X16 MOV ; T = X16 MPY P,XT,@M16 ; P = X16*M16 ; ACC = B32 MOVL ACC,@B32 ADDL ACC, P << PM ; ACC = ACC + (P >> 4) MOVI. @Y32.ACC : Store result into Y3 MOVL @Y32,ACC ; Store result into Y32

SQRA loc16 Square Value and Add P to ACC SYNTAX OPTIONS OPCODE OBJMODE RPT CYC SQRA loc16 Y 0101 0110 0001 0101 1 N+1 0000 0000 LLLL LLLL Addressing mode (see Chapter 5) Operands loc16 Description Add the previous product (stored in the P register), shifted by the amount specified by the product shift mode (PM), to the ACC register. Then the content of the location pointed to by the "loc16" addressing mode is loaded into the T register, squared, and stored in the P register: ACC = ACC + P << PM;T = [loc16];P = T * [loc16];Flags and Ζ After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. Modes Ν After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. С If the addition generates a carry, C is set; otherwise C is cleared. ν If an overflow occurs, V is set; otherwise V is not affected. ovc If overflow mode is disabled; and if the operation generates a positive overflow, then the counter is incremented. If overflow mode is disabled; and if the operation generates a negative overflow, then the counter is decremented. OVM If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFFFFFF) or maximum negative (0x80000000) if the operation overflowed. ΡM The value in the PM bits sets the shift mode for the output operation from the product register. If the product shift value is positive (logical left shift operation), then the low bits are zero filled. If the product shift value is negative (arithmetic right shift operation), the upper bits are sign extended. Repeat This instruction is repeatable. If the operation follows a RPT instruction, then it will be executed N+1 times. The state of the Z, N, C and OVC flags will reflect the final result. The V flag is set if an intermediate overflow occurs.

| Example | ; Calcul ; int16 ; sum = ; for(i= | ate sum of X[N] ; 0; 0; i < N; | f squares us Data inform i++) | ir at | ng 16-bit multiply: tion |
|---------|--|---|-------------------------------------|----------|--|
| | ; sum | = sum + | (X[i] * X[i] |) | >> 5; |
| | MOVL | XAR2,#X | | ; | XAR2 = pointer to X |
| | SPM | -5 | | ; | Set product shift to ">> 5" |
| | ZAPA | | | ; | Zero ACC, P, OVC |
| | RPT | #N-1 | | ; | Repeat next instruction N times |
| | SQRA | *XAR2++ | | ; ; | ACC = ACC + P >> 5, P = (*XAR2++)^2 |
| | ADDL | ACC,P << | PM | ; | Perform final accumulate |
| | MOVL | @sum,ACC | | ; | Store final result into sum |

SQRS loc16

Square Value and Subtract P From ACC

| | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|----------|---|--|---|--|---|
| SQRS loc16 | | | 0101 0110 0001 0001 xxxx xxxx LLLL LLLL | 1 | Y | N+1 |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Subtract the previous product (stored in the P register), shifted by the amount specified by the product shift mode (PM), from the ACC register. Then the content of the location pointed to by the "loc16" addressing mode is loaded into the T register, squared, and stored in the P register: ACC = ACC - P << PM; T = [loc16]; P = T * [loc16]; | | | | |
| Flags and Modes | Z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | |
| | Ν | After the subtraction cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | se N is |
| | С | If the subtraction ge | nerates a borrow, C is clear | red; otherwise | C is s | et. |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | |
| | OVC | If overflow mode is disabled; and if the operation generates a positive overflow, then the counter is incremented. If overflow mode is disabled; and if the operation generates a negative overflow, then the counter is decremented | | | | |
| | ΟνΜ | If overflow mode bi positive (0x7FFFFF overflowed. | it is set; then the ACC va FF) or maximum negative (0 | lue will satura x80000000) if | ate ma the op | ximum eration |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | bits sets the shift mode for the the product shift value is a low bits are zero filled. If right shift operation), the up | he output oper positive (log f the product oper bits are s | ation fr gical le shift va ign exte | om the ft shift alue is ended. |
| Repeat | | This instruction is rep it will be executed N reflect the final resu occurs. | peatable. If the operation foll I+1 times. The state of the ult. The V flag will be set i | lows a RPT in Z, N, C and (if an intermed | structio DVC fla liate ov | n, then ags will /erflow |

| Example | ; Calcul; ; int16 : ; sum = ; for(i= | ate sum of negativ X[N] ; Data in: 0; 0; i < N; i++) | ve squ format | uares using 16-bit multiply: tion |
|---------|---|---|------------------|--|
| | ; sum | = sum - (X[i] * 2 | X[i]) | >> 5; |
| | MOVL | XAR2,#X | ; | XAR2 = pointer to X |
| | SPM | -5 | ; | Set product shift to ">> 5" |
| | ZAPA | | ; | Zero ACC, P, OVC |
| | RPT | #N-1 | ; | Repeat next instruction N times |
| | SQRS | *XAR2++ | ; ; | ACC = ACC - P >> 5, $P = (*XAR2++)^{2}$ |
| | SUBL | ACC,P << PM | ; | Perform final subtraction |
| | MOVL | @sum,ACC | ; | Store final result into sum |

SUB ACC,loc16 << #0...16

Subtract Shifted Value From Accumulator

| S | ΥΝΤΑΧ Ο | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-----------|---|---|--|----------------------------------|------------------------------|--|
| SUB ACC, loc16 | ∂ << #0 | | 1010 1110 LLLL LLLL | 1 | Y | N+1 | |
| | | | 1000 0000 LLLL LLLL | 0 | - | 1 | |
| SUB ACC, loc16 | 6 << #115 | 5 | 0101 0110 0000 0000 | 1 | Y | N+1 | |
| | | | 0000 SHFT LLLL LLLL | | | | |
| | | | 1000 SHFT LLLL LLLL | 0 | - | 1 | |
| SUB ACC, loc16 | ∂ << #16 | | 0000 0100 LLLL LLLL | Х | Y | N+1 | |
| Operands | ACC | Accumulator registe | r | | | | |
| • | loc16 | Addressing mode (s | ee Chapter 5) | | | | |
| | #016 | Shift value (default i | s "<< #0" if no value specifi | ed) | | | |
| Description | | Subtract the left-shif mode from the ACC extension mode is tu (SXM= 0). The lowe if (SXM = 1) ACC = ACC - St else ACC = ACC - 0 | ted 16-bit location pointed t C register. The shifted value inned on (SXM=1) else the sl r bits of the shifted value an // sign extension m : [loc16] << shift value : [loc16] << shift value | o by the "loc1 ue is sign ext nifted value is re zero filled: mode enabled e; mode disable | 6" addr ended zero ex a | ressing if sign tended | |
| Flags and Modes | Z | After the subtraction | , the Z flag is set if ACC is | zero, else Z is | s cleare | ed. | |
| | Ν | After the subtractior cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, el: | se Z is | |
| | С | If the subtraction generation generation generation for a shift it. | nerates a borrow, C is clear of 16 is used, the SUB instru | red; otherwise uction can clea | e C is s r C but | et. not set | |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | | |
| | ovc | If(OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If(OVM = 1, enabled) then the counter is not affected by the operation. | | | | | |
| | SXM | If sign extension mo "loc16" field, will be s zero extended. | de bit is set; then the 16-bit sign extended before the add | operand, add dition. Else, th | ressed e value | by the will be | |
| | OVM | If overflow mode bi positive (0x7FFF F operation overflowed | t is set; then the ACC va FFF) or maximum negat d. | lue will satura ive (0x8000 | ate ma 0000) | ximum if the | |

| Repeat | If the operation is repeatable, then the instruction will be executed N+1 times. The state of the Z, N, C flags will reflect the final result. The V flag will be set if an intermediate overflow occurs. The OVC flag will count intermediate overflows, if overflow mode is disabled. If the operation is not repeatable, the instruction will execute only once. |
|---------|---|
| Example | ; Calculate signed value: ACC = (VarA << 10) - (VarB << 6); SETC SXM : Turn sign extension mode on |

| SETC | SXM | | | ; | Turn | sigr | ı exte | ensior | 1 mode | e on | | | |
|------|-----------|----|-----|---|-------|------|--------|--------|--------|-------|-----|----|------|
| MOV | ACC,@VarA | << | #10 | ; | Load | ACC | with | VarA | left | shift | ed | by | 10 |
| SUB | ACC,@VarB | << | #6 | ; | Subtr | act | VarB | left | shift | ed by | · 6 | to | ACC0 |

SUB ACC, loc16 <<T Subtract Shifted Value From Accumulator SYNTAX OPTIONS OPCODE OBJMODE RPT CYC SUB ACC,loc16 <<T Y 0101 0110 0010 0111 1 N+1 0000 0000 LLLL LLLL ACC Operands Accumulator register loc16 Addressing mode (see Chapter 5) Т Upper 16-bits of the multiplicand register, XT(31:16) Description Subtract from the ACC register the left-shifted contents of the 16-bit location pointed to by the "loc16" addressing mode. The shift value is specified by the four least significant bits of the T register, T(3:0) = shift value = 0..15. Higher order bits are ignored. The shifted value is sign extended if sign extension mode is turned on (SXM=1) else the shifted value is zero extended (SXM=0). The lower bits of the shifted value are zero filled: if(SXM = 1)// sign extension mode enabled ACC = ACC - S: [loc16] << T(3:0); // sign extension mode disabled else ACC = ACC - 0: [loc16] << T(3:0);Flags and Ζ After the subtraction, the Z flag is set if the ACC value is zero, else Z is Modes cleared. Ν After the subtraction, the N flag is set if bit 31 of the ACC is 1, else N is cleared. С If the subtraction generates a borrow, C is cleared; otherwise C is set. V If an overflow occurs, V is set; otherwise V is not affected. ovc If(OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. SXM If sign extension mode bit is set; then the 16-bit operand, addressed by the "loc16" field, will be sign extended before the addition. Else, the value will be zero extended. OVM If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFF FFFF) or maximum negative (0x8000 0000) if the operation overflowed. Repeat If this operation is repeated, then the instruction will be executed N+1 times. The state of the Z, N, C flags will reflect the final result. The V flag will be set if an intermediate overflow occurs. The OVC flag will count intermediate overflows, if overflow mode is disabled.

| Example | ; Cal | culate signed va | alue: | ACC = (VarA << SB) - (VarB << SB) |
|---------|-------|------------------|-------|--|
| | SETC | SXM | | ; Turn sign extension mode on |
| | MOV | T,@SA | | ; Load T with shift value in SA |
| | MOV | ACC,@VarA << T | | ; Load in ACC shifted contents of VarA |
| | MOV | T,@SB | | ; Load T with shift value in SB |
| | SUB | ACC,@VarB << T | | ; Subtract from ACC shifted contents ; of VarB |

SUB ACC,#16bit << #0..15

Subtract Shifted Value From Accumulator

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------------|---------------------|---------|-----|-----|
| SUB ACC,#16bit << #015 | 1111 1111 0000 SHFT | Х | - | 1 |

| Operands | ACC | Accumulator register |
|--------------------|------------------------------|---|
| | #16bit | 16-bit immediate constant value |
| | #015 | Shift value (default is "<< #0" if no value specified) |
| Description | | Subtract the left shifted 16-bit immediate constant value from the ACC register. The shifted value is sign extended if sign extension mode is turned on (SXM=1) else the shifted value is zero extended (SXM=0). The lower bits of the shifted value are zero filled: if (SXM = 1) // sign extension mode enabled ACC = ACC - S:16bit << shift value; else // sign extension mode disabled |
| | | ACC = ACC - 0:16bit << shift value; |
| | | Smart Encoding: If #16bit is an 8-bit number and the shift is zero, then the assembler will encode this instruction as SUBB ACC, #8bit for improved efficiency. To override this encoding, use the SUBW ACC, #16bit instruction alias. |
| Flags and Modes | Z N C | After the subtraction, the Z flag is set if ACC is zero, else Z is cleared. After the subtraction, the N flag is set if bit 31 of the ACC is 1, N is cleared. If the subtraction generates a borrow, C is cleared; otherwise C is set. |
| | V | If an overflow occurs, V is set; otherwise V is not affected. |
| | ovc | If (OVM = 0, disabled) then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If (OVM = 1, enabled) then the counter is not affected by the operation. |
| | SXM | If sign extension mode bit is set; then the 16-bit operand, addressed by the "loc16" field, will be sign extended before the addition. Else, the value will be zero extended |
| | OVM | If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFFFFFF) or maximum negative (0x80000000) if the operation overflowed. |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. |
| Example | ; Calc SETC MOV SUB | culate signed value: ACC = (VarB << 10) - (23 << 6);SXM; Turn sign extension mode onACC,@VarB << #10; Load ACC with VarB left shifted by 10ACC,#23 << #6; Subtract from ACC 23 left shifted by 6 |

SUB AX, loc16

Subtract Specified Location From AX

| 5 | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|------------------------------|--|---|--|--------------------|-----|--|--|--|
| SUB AX, loc16 | | | 1001 111A LLLL LLLL | Х | - | 1 | | | |
| Operands | AX loc16 | Accumulator high (A Addressing mode (s | ccumulator high (AH) or accumulator low (AL) register Idressing mode (see Chapter 5) | | | | | | |
| Description | | Subtract the 16-bit content of the location pointed to by the "loc16" addressing mode from the specified AX register (AH or AL) and store the results in AX: AX = AX - [loc16]; | | | | | | | |
| Flags and Modes | Ν | After the subtraction, then the negative fla | fter the subtraction, AX is tested for a negative condition. If bit 15 of AX is 1, ien the negative flag bit is set; otherwise it is cleared. | | | | | | |
| | Z | After the subtraction, if the operation gene | After the subtraction, AX is tested for a zero condition. The zero flag bit is set f the operation generates $AX = 0$, otherwise it is cleared | | | | | | |
| | С | If the subtraction ge | nerates a borrow, C is clear | red; otherwise | e C is s | et. | | | |
| | V | If an overflow occurs overflow occurs if the positive direction. Si max negative value | If an overflow occurs, V is set; otherwise V is not affected. Signed positive overflow occurs if the result crosses the max positive value (0x7FFF) in the positive direction. Signed negative overflow occurs if the result crosses the max negative value (0x8000) in the negative direction. | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Subti MOV SUB MOV | ract the contents o AL,@VarA AL,@VarB @VarC,AL | of VarA with VarB and s ; Load AL with cont ; Subtract from AL ; Store result in V | store in Va cents of Van contents of VarC | rC rA E VarB | | | | |

SUB loc16, AX

Reverse-Subtract Specified Location From AX

| | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|--|--|--|--|-------------------------------|---------------------------------|--|--|--|
| SUB loc16, AX | | | 0111 010A LLLL LLLL | Х | - | 1 | | | |
| Operands | loc16 AX | Addressing mode (s Accumulator high (A | dressing mode (see Chapter 5) cumulator high (AH) or accumulator low (AL) register | | | | | | |
| Description | | Subtract the content content of the location the result in location [loc16] = [loc16] | of the specified AX registe on pointed to by the "loc16" pointed to by "loc16": AX; | r (AH or AL) fi addressing m | rom the ode an | e 16-bit d store | | | |
| Flags and Modes | Ν | After the subtractior [loc16] is 1, then the negati | n, [loc16] is tested for a neg | gative conditionities is cleared. | on. If bi | it 15 of | | | |
| | Z | After the subtraction set if the operation | fter the subtraction, [loc16] is tested for a zero condition. The zero flag bit is et if the operation generates $[loc16] = 0$; otherwise it is cleared | | | | | | |
| | С | If the subtraction ge | nerates a borrow, C is clea | red; otherwise | e C is s | et. | | | |
| | v | If an overflow occurs overflow occurs if th positive direction. Si max negative value | s, V is set; otherwise V is n e result crosses the max po gned negative overflow occ (0x8000) in the negative di | ot affected. Si sitive value (C curs if the resu rection. | igned p x7FFF ilt cross | oositive) in the ses the | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only c | e RPT once. | | | |
| Example | ; Subt: MOV SUB ; Subt: MOV SUB | ract the contents AL,@VarA @AR0,AL ract the contents AH,@VarB @VarC,AH | of VarA from index reg ; Load AL with cont ; ARO = ARO - AL of VarB from VarC: ; Load AH with cont ; VarC = VarC - AH | ister ARO: cents of Var cents of Var | cA cB | | | | |

SUBB ACC,#8bit

Subtract 8-bit Value

| S | YNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|--------------------------------|--|--|---|---------------------|---------------|--|--|
| SUBB ACC,#8b | oit | | 0001 1001 CCCC CCCC | Х | - | 1 | | |
| Operands | ACC #8bit | Accumulator register 8-bit immediate cons | ccumulator register bit immediate constant value | | | | | |
| Description | | Subtract the zero-ex ACC = ACC - 0:8bi | ubtract the zero-extended, 8-bit constant from the ACC register: CC = ACC - 0:8bit; | | | | | |
| Flags and Modes | Z N C | After the subtraction After the subtraction If the subtraction get | er the subtraction, the Z flag is set if ACC is zero, else Z is cleared. ter the subtraction, the N flag is set if bit 31 of the ACC is 1, N is cleared. he subtraction generates a borrow, C is cleared; otherwise C is set. | | | | | |
| | v | If an overflow occurs | It an overtiow occurs, V is set; otherwise, V is not affected. | | | | | |
| | ovc | If(OVM = 0, disabled then the counter is in overflow, then the co counter is not affected | If $(OVM = 0, disabled)$ then if the operation generates a positive overflow, then the counter is incremented and if the operation generates a negative overflow, then the counter is decremented. If $(OVM = 1, enabled)$ then the counter is not affected by the operation | | | | | |
| | OVM | If overflow mode bi positive (0x7FFFFF overflowed. | overflow mode bit is set; then the ACC value will saturate maximum ositive (0x7FFFFFFF) or maximum negative (0x80000000) if the operation verflowed. | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | ⇒ RPT nce. | | |
| Example | ; Dec: MOVL SUBB MOVL | rement contents c ACC,@VarA ACC,#1 @VarA,ACC | f 32-bit location Va ; Load ACC with cont ; Subtract 1 from AC ; Store result back | arA: tents of Va: CC into VarA | rA | | | |

SUBB SP,#7bit

| S | YNTAX OF | TIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|-------------|--|------------|---|----------------------------------|---------------------|----------------|
| SUBB SP,#7bit | | | 1111 | 1110 1CCC CCCC | Х | - | 1 |
| Operands | SP #7bit | Stack pointer 7-bit immediate con | stant v | /alue | | | |
| Description | | Subtract a 7-bit uns SP = SP - 0:7bit | igned ; | constant to SP and st | ore the result | in SP: | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is instruction, it resets | not re | epeatable. If this ins peat counter (RPTC) | struction follo and executes | ows the s only o | ∍ RPT ince. |
| Example | FuncA: | | ; | Function with loc stack. | al variable | es on | |
| | ADI | DB SP,#N | ; ; | Reserve N 16-bit local variables c | words of s <u>p</u> on stack: | bace f | or |
| | • | | | | | | |
| | SUE LRE | BB SP,#N STR | ; ; | Deallocate reserv Return from funct | red stack sp zion. | bace. | |

SUBB XARn,#7bit

Subtract 7-Bit From Auxiliary Register

| S | YNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------|--|--|---------------------------------|------------------|---------------|
| SUBB XARn, # | 7bit | | 1101 1nnn 1CCC CCCC | Х | - | 1 |
| Operands | XARr #7bit | XAR0 to XAR7, 32-b 7-bit immediate con | bit auxiliary registers stant value | | | |
| Description | | Subtract the 7-bit u XARn: | insigned constant from XA | Rn and store | the re | sult in |
| | | XARn = XARn - 0:7 | /bit; | | | |
| Flags and Modes | | None | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |
| Example | MOVL | XAR1,#VarA | ; Initialize XAR1 <u>p</u> ; of VarA | oointer with | n addr | ess |
| | MOVL | XAR2,*XAR1 | ; Load XAR2 with co | ontents of N | VarA | |
| | SUBB | XAR2,#10h` | ; XAR2 = VarA - 0x1 | L 0 | | |

SUBBL ACC, loc32

Subtract 32-bit Value Plus Inverse Borrow

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|--|---------|-----|-----|
| SUBBL ACC, loc32 | 0101 0110 0101 0100 0000 0000 LLLL LLLL | 1 | - | 1 |

| Operands | loc32 | loc32 Addressing mode (see Chapter 5) | | | | | |
|--------------------|--------------------|--|--|--|--|--|--|
| | ACC | Accumulator register | | | | | |
| Description | | Subtract from the ACC addressing mode and the | C the 32-bit location pointed to by the "loc32" le logical inversion of the value in the carry flag bit: | | | | |
| | | ACC = ACC - [loc32] | - ~C; | | | | |
| Flags and Modes | Z | After the subtraction, th cleared. | e Z flag is set if the ACC value is zero, else Z is | | | | |
| | Ν | After the subtraction, th cleared. | e N flag is set if bit 31 of the ACC is 1, else N is | | | | |
| | С | The state of the carry bit subtraction generates a | before execution is included in the subtraction. If the borrow, C is cleared; otherwise C is set. | | | | |
| | V | If an overflow occurs, V is set; otherwise V is not affected. | | | | | |
| | ovc | If(OVM = 0, disabled) the then the counter is incre- overflow, then the count counter is not affected b | then if the operation generates a positive overflow, emented and if the operation generates a negative ser is decremented. If $(OVM = 1, enabled)$ then the y the operation. | | | | |
| | OVM | If overflow mode bit is positive (0x7FFFFFFF) overflowed. | set; then the ACC value will saturate maximum or maximum negative (0x80000000) if the operation | | | | |
| Repeat | | This instruction is not instruction, it resets the | repeatable. If this instruction follows the RPT repeat counter (RPTC) and executes only once. | | | | |
| Example | ; Subtı ; in Va | ract two 64-bit value: arC: | s (VarA and VarB) and store result | | | | |
| | MOVL | ACC,@VarA+0 | ; Load ACC with contents of the low ; 32-bits of VarA | | | | |
| | SUBUL | ACC,@VarB+0 | ; Subtract from ACC the contents of ; the low 32-bits of VarB | | | | |
| | MOVL | <pre>@VarC+0,ACC</pre> | ; Store low 32-bit result into VarC | | | | |
| | MOVL | ACC,@VarA+2 | ; Load ACC with contents of the high ; 32-bits of VarA | | | | |

| SUBBL | ACC,@VarB+2 | ; | Subtract | fro | m ACC | the | cor | ntents | s of |
|-------|------------------------|---|-----------|------|-------|-------|-----|--------|--------|
| | | ; | the high | 32- | bits | of Va | arB | with | borrow |
| MOVL | <pre>@VarC+2,ACC</pre> | ; | Store hig | gh 3 | 2-bit | res | ult | into | VarC |

SUBCU ACC, loc16

Subtract Conditional 16 Bits

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|--|--|--|---|
| SUBCU ACC,loc16 | | | 0001 1111 LLLL LLLL | Х | Y | N+1 |
| Operands | ACC loc32 | Accumulator register Addressing mode (see Chapter 5) | | | | |
| Description | | Perform 16-bit conditional subtraction, which can be used for unsignem modulus division: | | | | |
| | | <pre>temp(32:0) = ACC << 1 - [loc16] << 16 if(temp(32:0) >= 0) ACC = temp(31:0) + 1 else ACC = ACC << 1</pre> | | | | |
| | | To perform 16-bit un the AL register is los SUBCU instruction. contains the "Denon times, the AH regist contain the "Quotie "Numerator" and "E quantities, before e result must be negat different sign else th | signed modulus division, aded with the "Numerato The value pointed to be ninator" value. After exect er will contain the "Rema nt" results. To perform Denominator" values mu xecuting the SUBCU in ted if the "Numerator" and he quotient is left unchan | the AH register r" value prior to the "loc16" add inder" and the A signed modulus st be converted struction. The fi "Denominator" ged. | is zero execut ressing l instruct AL regis d to un nal "Qu values | ed and ing the mode tion 16 ster will on, the signed uotient" were of |
| Flags and Modes | Z | At the end of the ope cleared. The calcula | eration, the Z flag is set if t ation of temp(32:0) has n | he ACC value is o effect on the Z | zero, e 1 bit. | lse Z is |
| | Ν | At the end of the ope cleared. The calcula | eration, the N flag is set if ation of temp(32:0) has n | bit 31 of the ACC o effect on the N |) is 1, e I bit. | lse N is |
| | С | If the calculation of temp(32:0) generates a borrow, C is cleared; otherwise C is set. | | | | rwise C |
| | | Note: The V and OVC | flags are not affected by the o | peration. | | |
| Repeat | | If this operation is re The state of the Z, N an intermediate over overflows, if overflow | peated, then the instructi , C flags will reflect the fin erflow occurs. The OVC w mode is disabled. | on will be execut al result. The V fl flag will count | ed N+1 ag will k t intern | times. be set if nediate |
| Example 1 | ; Calcu MOVU RPT SUBCU MOV MOV | llate unsigned: Qu ACC,@Num16 #15 J ACC,@Den16 @Rem16,AH @Ouot16,AL | ot16 = Num16Den16, R ; AL = Num16, ; Repeat opera ; Conditional ; Store remain ; Store quotie | eml6 = Numl6% AH = 0 tion 16 times subtract with der in Rem16 nt in Ouot16 | Den16 Den16 | ; |

| Example 2 | ; Calcula | te signed: Quot16 = | Num16Den16, Rem16 = Num16%Den16 |
|-----------|-----------|---------------------|--|
| - | CLRC | ТС | ; Clear TC flag, used as sign flag |
| | MOV | ACC,@Den16 << 16 | ; AH = Den16, AL = 0 |
| | ABSTC | ACC | ; Take abs value, TC = sign ^ TC |
| | MOV | Т,@АН | ; Temp save Den16 in T register |
| | MOV | ACC,@Num16 << 16 | ; $AH = Num16$, $AL = 0$ |
| | ABSTC | ACC | ; Take abs value, TC = sign ^ TC |
| | MOVU | ACC,@AH | ; $AH = 0$, $AL = Num16$ |
| | RPT | #15 | ; Repeat operation 16 times |
| | SUBCU | ACC,@T | ; Conditional subtract with Den16 |
| | MOV | @Rem16,AH | ; Store remainder in Rem16 |
| | MOV | ACC,@AL << 16 | ; AH = Quot16, AL = 0 |
| | NEGTC | ACC | ; Negate if TC = 1 |
| | MOV | @Quot16,AH | ; Store quotient in Quot16 |
| Example 3 | ; Calcula | te unsigned: Quot32 | = Num32/Den16, Rem16 = Num32%Den16 |
| | MOVU | ACC,@Num32+1 | ; AH = 0, AL = high 16-bits of Num32 |
| | RPT | #15 | ; Repeat operation 16 times |
| | SUBCU | ACC,@Den16 | ; Conditional subtract with Den16 |
| | MOV | @Quot32+1,AL | ; Store high 16-bit in Quot32 |
| | MOV | AL,@Num32+0 | ; AL = low 16-bits of Num32 |
| | RPT | #15 | ; Repeat operation 16 times |
| | SUBCU | ACC,@Den16 | ; Conditional subtract with Den16 |
| | MOV | @Rem16,AH | ; Store remainder in Rem16 |
| | MOV | @Quot32+0,AL | ; Store low 16-bit in Quot32 |
| Example 4 | ; Calcula | te signed: Ouot32 = | Num32/Den16, Rem16 = Num32%Den16 |
| | CLRC | TC ~ | : Clear TC flag, used as sign flag |
| | MOV | ACC.@Den16 << 16 | : $AH = Den16$. $AL = 0$ |
| | ABSTC | ACC | \cdot Take abs value TC = sign TC |
| | MOV | т @дн | · Temp save Den16 in T register |
| | MOVI | ACC @Num22 | , remp bave benefit in r regibeer $\sqrt{2C} = N_{1}m^{2}$ |
| | ADGTC | ACC, ENGINEZ | , $Acc = Namsz$ |
| | MOV | ACC D @ACC | , Take abs value, IC - Sign IC |
| | MOV | P, WACC | r = Null 2 |
| | MOVU | ACC, @PH | ; AH = 0, AL = HIGH 16-DICS OF NUM32 |
| | RPT | #15 NGC 07 | ; Repeat operation 16 times |
| | ISOBCO | ACC, @T | ; Conditional subtract with Denie |
| | MOV | @Quot32+1,AL | ; Store high 16-bit in Quot32 |
| | MOV | AL,@PL | ; AL = low 16-bits of Num32 |
| | RPT | #15 | ; Repeat operation 16 times |
| | SUBCU | ACC,@T | ; Conditional subtract with Den16 |
| | MOV | @Rem16,AH | ; Store remainder in Rem16 |
| | MOV | ACC,@AL << 16 | ; AH = low 16-bits of Quot32, AL = 0 |
| | NEGTC | ACC | ; Negate if TC = 1 |
| | MOV | @Quot32+0,AH | ; Store low 16-bit in Quot32 |

SUBCUL ACC, loc32

Subtract Conditional 32 Bits

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|------------------|--|---------|-----|-----|
| SUBCUL ACC,loc32 | 0101 0110 0001 0111 0000 0000 LLLL LLLL | 1 | Y | N+1 |

Operands ACC Accumulator register

loc32 Addressing mode (see Chapter 5)

Description Perform 32-bit conditional subtraction, which can be used for unsigned modulus division:

```
temp(32:0) = ACC << 1 + P(31) - [loc32];
if( temp(32:0) >= 0 )
   ACC = temp(31:0);
   P = (P << 1) + 1;
else
   ACC:P = ACC:P << 1;</pre>
```

To perform 32-bit unsigned modulus division, the ACC register is zeroed and the P register is loaded with the "Numerator" value prior to executing the SUBCUL instruction. The value pointed to be the "loc32" addressing mode contains the "Denominator" value. After executing the SUBCUL instruction 32 times, the ACC register will contain the "Remainder" and the P register will contain the "Quotient" results. To perform signed modulus division, the "Numerator" and "Denominator" values must be converted to unsigned quantities, before executing the SUBCUL instruction. The final "Quotient" result must be negated if the "Numerator" and "Denominator" values were of different sign else the quotient is left unchanged.

| Flags and Modes | Z | At the end of the operation, the Z flag is set if the ACC value is zero, else Z is cleared. The calculation of temp(32:0) has no effect on the Z bit. |
|--------------------|---|---|
| | Ν | At the end of the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared. The calculation of temp(32:0) has no effect on the N bit. |
| | С | If the calculation of temp(32:0) generates a borrow, C is cleared; otherwise C is set. |
| | | Note: The V and OVC flags are not affected by the operation. |
| Repeat | | If this operation is repeated, then the instruction will be executed N+1 times. The state of the Z, N, C flags will reflect the final result. The V flag will be set if an intermediate overflow occurs. The OVC flag will count intermediate overflows, if overflow mode is disabled. |
| Example 1 | ; Calcula | te unsigned: | Quot32 = Num32/Den32, Rem32 = Num32%Den32 |
|-----------|-----------|--------------|---|
| - | MOVB | ACC,#0 | ; Zero ACC |
| | MOVL | P,@Num32 | ; Load P register with Num32 |
| | RPT | #31 | ; Repeat operation 32 times |
| | SUBCUL | ACC,@Den32 | ; Conditional subtract with Den32 |
| | MOVL | @Rem32,ACC | ; Store remainder in Rem32 |
| | MOVL | @Quot32,P | ; Store quotient in Quot32 |
| Example 2 | ; Calcula | te signed: Q | uot32 = Num32/Den32, Rem32 = Num32%Den32 |
| | CLRC | TC | ; Clear TC flag, used as sign flag |
| | MOVL | ACC,@Den32 | ; Load ACC with contents of Den32 |
| | ABSTC | ACC | ; Take absolute value, TC = sign ^ TC |
| | MOVL | XT,@ACC | ; Temp save denominator in XT register |
| | MOVL | ACC,@Num32 | ; Load ACC register with Num32 |
| | ABSTC | ACC | ; Take abs value, TC = sign ^ TC |
| | MOVL | P,@ACC | ; Load P register with numerator |
| | MOVB | ACC,#0 | ; Zero ACC |
| | RPT | #31 | ; Repeat operation 32 times |
| | SUBCUL | ACC,@XT | ; Conditional subtract with denominator |
| | MOVL | @Rem32,ACC | ; Store remainder in Rem32 |
| | MOVL | ACC,@P | ; Load ACC with quotient |
| | NEGTC | ACC | ; Negate ACC if TC=1 (negative result) |
| | MOVL | @Quot32,ACC | ; Store quotient in Quot32 |
| Example 3 | ; Calcula | te unsigned: | Quot64 = Num64Den32, Rem32 = Num64%Den32 |
| | MOVB | ACC,#0 | ; Zero ACC |
| | MOVL | P,@Num64+2 | ; Load P with high 32-bits of Num64 |
| | RPT | #31 | ; Repeat operation 32 times |
| | SUBCUL | ACC,@Den32 | ; Conditional subtract with Den32 |
| | MOVL | @Quot64+2,P | ; Store high 32 bit quotient in Quot64 |
| | MOVL | P,@Num64+0 | ; Load P with low 32-bits of Num64 |
| | RPT | #31 | ; Repeat operation 32 times |
| | SUBCUL | ACC,@Den32 | ; Conditional subtract with Den32 |
| | MOVL | @Rem32,ACC | ; Store remainder in Rem32 |
| | MOVL | @Quot64+0,P | ; Store low 32 bit quotient in Quot64 |

| Example 4 | ; Calcula | te signed: Quot64 | = : | Num364Den32, Rem32 = Num64%Den32 |
|-----------|-----------|-------------------|-----|---------------------------------------|
| | MOVL | ACC,@Num64+2 | ; | Load ACC:P with 64-bit numerator |
| | MOVL | P,@Num64+0 | | |
| | TBIT | @AH,#15 | ; | TC = sign of numerator |
| | SBF | \$10,NTC | ; | Take absolute value of numerator |
| | NEG64 | ACC:P | | |
| | \$10: | | | |
| | MOVL | @XAR3,P | ; | Temp save numerator low in XAR3 |
| | MOVL | P,@ACC | ; | Load P register with numerator high |
| | MOVL | ACC,@Den32 | ; | Load ACC with contents of Den32 |
| | ABSTC | ACC | ; | Take absolute value, TC = sign ^ TC |
| | MOVL | XT,@ACC | ; | Temp save denominator in XT register |
| | MOVB | ACC,#0 | ; | Zero ACC |
| | RPT | #31 | ; | Repeat operation 32 times |
| | SUBCUL | ACC,@XT | ; | Conditional subtract with denominator |
| | MOVL | @XAR4,P | ; | Store high quotient in XAR4 |
| | MOVL | P,@XAR3 | ; | Load P with low numerator |
| | RPT | #31 | ; | Repeat operation 32 times |
| | SUBCUL | ACC,@XT | ; | Conditional subtract with denominator |
| | MOVL | @Rem32,ACC | ; | Store remainder in Rem32 |
| | MOVL | ACC,@XAR4 | ; | Load ACC with high quotient from XAR4 |
| | SBF | \$20,NTC | ; | Take absolute value of quotient |
| | NEG64 | ACC:P | | |
| | \$20: | | | |
| | MOVL | @Quot64+0,P | ; | Store low quotient into Quot64 |
| | MOVL | @Quot64+2,ACC | ; | Store high quotient into Quot64 |

SUBL ACC, loc32

Subtract 32-bit Value

| 5 | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------|--|---|----------------------------------|-------------------------------|--------------------|
| SUBL ACC, loc | :32 | | 0000 0011 LLLL LLLL | 1 | - | 1 |
| Operands | ACC | Accumulator registe | r | | | |
| | loc32 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Subtract the 32-bit lot the ACC register : | ocation pointed to by the "lo | c32" addressi | ng mod | le from |
| | | ACC = ACC - [loc3] | 32]; | | | |
| Flags and Modes | z | After the subtractior cleared. | n, the Z flag is set if the A | CC value is z | ero, els | se Z is |
| | Ν | After the subtractior cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | se N is |
| | С | If the subtraction ge | nerates a borrow, C is clea | red; otherwise | e C is s | et. |
| | V | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | |
| | ovc | If OVM = 0 (disabled then the counter is i overflow, then the co | d), then if the operation ger ncremented and if the oper ounter is decremented. | nerates a pos ration generat | itive ov es a ne | erflow, egative |
| | | If OVM = 1 (enabled | l), then the counter is not at | fected by the | operat | ion. |
| | ΟνΜ | If overflow mode bi positive (0x7FFFFF overflowed. | t is set; then the ACC va FF) or maximum negative (0 | lue will satura x80000000) if | ate ma [:] the op | ximum eration |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT ince. |
| Example | ; Calc | culate the 32-bit va | alue: VarC = VarA-VarB | | | |
| | MOVL | ACC,@VarA | ; Load ACC with cor | ntents of Va | arA | |
| | SUBL | ACC,@VarB | ; Subtract from ACC | C the conter | nts of | VarB |
| | MOVL | @VarC,ACC | ; Store result into | o VarC | | |

SUBL ACC,P << PM

Subtract 32-bit Value

| 5 | SYNTAX | OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|---|---|--|---|--|---|
| SUBL ACC,P < | << PM | | 0001 | 0001 1010 1100 | Х | Y | N+1 |
| Note This instr | uction is a | n alias for the "MOVS T loc1 | 6" opera | tion with "loc16 = $@T$ " a | ddressing mode | | |
| Operende | | | | | | | |
| Operands | ACC | Accumulator registe | er | | | | |
| | Р | Product register | | | | | |
| | < <pm< td=""><td>Product shift mode</td><td></td><td></td><td></td><td></td><td></td></pm<> | Product shift mode | | | | | |
| Description | | Subtract the content mode (PM), from th | t of the le conte | P register, shifted as ent of the ACC regis | specified by th ster: | e produ | ıct shift |
| | | ACC = ACC - P << | PM; | | | | |
| Flags and Modes | z | After the subtractio cleared. | n, the l | Z flag is set if the A | ACC value is z | ero, el | se Z is |
| | Ν | After the subtractio cleared. | n, the I | N flag is set if bit 3 | 1 of the ACC i | s 1, els | se N is |
| | С | If the subtraction ge | enerate | s a borrow, C is cle | ared; otherwise | e C is s | et. |
| | v | If an overflow occur | rs, V is | set; otherwise V is | not affected. | | |
| | OVC | If OVM = 0 (disable counter is incremen counter is decremen If OVM = 1 (enabled | d) and nted; if t nted. d), the | the operation gene he operation gener counter is not affect | rates a positive ates a negative red by the oper | e overfle e overfle ation. | ow, the ow, the |
| | OVM | If overflow mode b positive (0x7FFFF overflowed. | it is se FF) or i | et; then the ACC v maximum negative (| alue will satur (0x80000000) it | ate ma the op | ximum eration |
| | РМ | The value in the PM product register. If operation), then the negative (arithmetic | l bits se the pi e low k right s | ts the shift mode for roduct shift value bits are zero filled. hift operation), the u | the output open is positive (log If the product upper bits are s | ration fr gical le shift v sign ext | om the ft shift alue is ended. |
| Repeat | | If this operation is re The state of the Z, N an intermediate ov overflows, if overflo | epeatec I, C flag verflow w mode | l, then the instruction s will reflect the final occurs. The OVC e is disabled. | n will be execut result. The V fla flag will count | ed N+1 ag will t interm | times. De set if Dediate |
| Example | ; Calo ; Y, N | culate: Y = ((B << 4, X, B are Q15 val | 11) - lues | (M*X >> 4)) >> | 10 | | |
| | SPM | -4 | ; | Set product shi: | Et to >> 4 | | |
| | SETC | SXM | ; | Enable sign exte | ension mode | | |
| | MOV | T,@M D T @Y | ; | T = M D = M * V | | | |
| | MOV | r, r, w_A ACC @B << 11 | ; | $r = m - \Lambda$ $\Lambda CC = S \cdot B < 11$ | | | |

 SUBL
 ACC, P << PM</th>
 ; ACC = (S:B << 11) - (M*X >> 4)

 MOVH
 @Y,ACC << 5</td>
 ; Store Q15 result into Y

SUBL loc32, ACC

Subtract 32-bit Value

| | SYNTAX | OPTIONS | | OP | CODE | | OBJMODE | RPT | CYC |
|--------------------|--------|---|--|------------------------------------|-----------------------------|---------------------------------------|---|---------------------------------|--------------------|
| SUBL loc32, A | CC | | 0101 0000 | 0110 0000 | 0100 LLLL | 0001 LLLL | 1 | - | 1 |
| Operands | loc32 | Addressing mode (s | see Cha | apter | 5) | | | | |
| | ACC | Accumulator registe | er | | | | | | |
| Description | | Subtract the content "loc32" addressing r | t of the mode: | ACC | registe | er from th | ne location poi | nted to | by the |
| | | [loc32] = [loc32 |] – AC | 2C; | | | | | |
| Flags and Modes | Z | After the subtraction cleared. | n, the 2 | Z flag | is set | if the A | CC value is z | ero, el | se Z is |
| | Ν | After the subtractior cleared. | n, the N | l flag | is set | if bit 31 d | of the [loc32] | is 1, els | se N is |
| | С | If the subtraction ge | enerate | s a bo | orrow, | C is clea | red; otherwise | e C is s | et. |
| | v | If an overflow occur | s, V is | set; o | therwi | se V is n | ot affected. | | |
| | OVC | If OVM = 0 (disable counter is incremen the counter is decre If OVM = 1 (enabled | d) and ted and emented d) the c | the op d if the d. counte | peratic opera r is no | on genera ation gen ot affected | ates a positive lerates a nega d by the opera | e overflo ative ov ation. | ow, the erflow, |
| | OVM | If overflow mode b positive (0x7FFFFI overflowed. | it is se FF) or r | et; the maxim | n the ium ne | ACC va egative (0 | lue will satura x80000000) if | ate ma ^t the op | ximum eration |
| Repeat | | This instruction is instruction, it resets | not re the rep | epeata beat c | ıble. I ountei | f this in r (RPTC) | struction follo and executes | ows the s only c | e RPT |
| Example | ; Dec | rement the 32-bit v | alue N | /arA: | | | | | |
| | MOVB | ACC,#1 | ; | Load | ACC | with Ox(| 0000001 | | |
| | SUBL | @VarA,ACC | ; | VarA | = Va | rA – ACO | 2 | | |

SUBR loc16,AX

Reverse-Subtract Specified Location From AX

| 5 | Ο ΧΑΤΑΥ | PTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---|--|--|--|---------------------------------------|------------------------------|
| SUBR loc16,AX | < | | 1110 101A LLLL LLLL | 1 | - | 1 |
| Operands | loc16 AX | Addressing mode (s Accumulator high (A | ee Chapter 5) H) or accumulator low (AL) | register | | |
| Description | | Subtract the 16-bit addressing mode fro result in location poi | t content of the location om the specified AX registe nted to by "loc16": | pointed to by r (AH or AL), | y the ' and sto | "loc16" ore the |
| | | [loc16] = AX - [] | loc16] | | | |
| | | This instruction perfe | orms a read-modify-write op | peration. | | |
| Flags and Modes | Ν | After the subtraction [loc16] is 1, then the | n, [loc16] is tested for a neg negative flag bit is set; oth | gative condition erwise it is cle | on. If bi eared. | it 15 of |
| | Z | After the subtraction bit is set if the opera | , [loc16] is tested for a zero tion generates [loc16] = 0, | o condition. Th otherwise it is | ne zero cleare | flag ed |
| | С | If the subtraction ge | nerates a borrow, C is clea | red; otherwise | e C is s | et. |
| | v | If an overflow occurs overflow occurs if th the positive direction crosses the max neg | s, V is set; otherwise V is no e result crosses the max po n. Signed negative overflow gative value (0x8000) in the | ot affected. Si ositive value (occurs if the e negative dire | gned p 0x7FFf result ection. | ositive ⁻) in |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT once. |
| Example | ; Subtr MOV sign ex with a SUBR ; Subtr MOV SUBR | ract index register AL,@VarA ; Enable tensio ; left shift of 3 @AR0,AL ract the contents of AH,@VarB @VarC,AH | r AR0 from VarA and sto ; Load AL with cont ; AR0 = AL - AR0 of VarC from VarB and ; ; Load AH with cont ; VarC = AH - VarC | ore in ARO: cents of Van store in Va cents of Van | rC: rB | |

SUBRL loc32, ACC

Reverse-Subtract Specified Location From ACC

| : | SYNTAX C | OPTIONS | | OF | CODE | | OBJMODE | RPT | CYC |
|--------------------|------------------|---|---------------------------------------|--|--|---|--|---------------------------------|--------------------------------|
| SUBRL loc32, | ACC | | 0101 0000 | 0110 | 0100 LLLL | 1001 LLLL | 1 | - | 1 |
| Operands | loc32 | Addressing mode (s | see Cł | napter | 5) | | | | |
| | ACC | Accumulator registe | er | | | | | | |
| Description | | Subtract from the A addressing mode at [loc32] = ACC - | CC real nd sto | gister re the 2] ; | the 32- result | bit location bit location in the loc | on pointed to ation pointed | by the to by " | "loc32" loc32": |
| Flags and Modes | Z | After the subtractio cleared. | n, the | Z flag | ı is set | if the A | CC value is z | ero, el | se Z is |
| | Ν | After the subtractio cleared. | n, the | N flag | j is sei | t if bit 31 | of the ACC i | s 1, els | ₃e N is |
| | С | If the subtraction ge | enerate | es a bo | orrow, | C is clea | red; otherwise | e C is s | et. |
| | V | If an overflow occur | rs, V is | set; c | otherwi | se V is n | ot affected. | | |
| | OVC | If(OVM = 0, disable then the counter is overflow, then the c counter is not affect | ed) the increm counte ted by | n if th nented r is de the op | e oper and if creme peratio | ation ger the oper nted. If(C n. | nerates a pos ration generat DVM = 1, enal | itive ov es a ne bled) th | ′erflow, ∋gative ìen the |
| | ΟνΜ | If overflow mode b positive (0x7FFFFF overflowed. | it is s FF) or | et; the maxin | en the num ne | ACC va egative (0 | lue will satur x80000000) if | ate ma the op | ximum eration |
| Repeat | | This instruction is instruction, it resets | not r the re | epeat epeat o | able. I counte | f this in r (RPTC) | struction follo and executes | ows the | ∍ RPT once. |
| Example | ; Calc MOVL . | ulate the 32-bit v ACC,@VarB | value: ; | VarA Load | A = Va ACC | rB - Va with com | rA ntents of Va | arB | |
| | SUBRL | @VarA,ACC | ; | VarA | = AC | C – Vari | Α | | |

SUBU ACC, loc16

Subtract Unsigned 16-bit Value

| 5 | SYNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--------------|---|--|--|---------------------------------|--------------------------------|
| SUBU ACC, lo | c16 | | 0000 0001 LLLL LLLL | Х | Y | N+1 |
| Operands | ACC loc16 | Accumulator registe Addressing mode (s | r ee Chapter 5) | | | |
| Description | | Subtract the 16-bit addressing mode fr extended before the | contents of the location om the ACC register. The add: | pointed to b addressed lo | y the cation | "loc16" is zero |
| | | ACC = ACC - 0: []c | pc16]; | | | |
| Flags and Modes | Z | After the subtraction | i, the Z flag is set if ACC is | zero, else Z is | s cleare | ∍d. |
| | Ν | After the subtractior cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | se N is |
| | С | If the subtraction ge | nerates a borrow, C is clea | red; otherwise | e C is s | et. |
| | v | If an overflow occurs | s, V is set; otherwise V is n | ot affected. | | |
| | OVC | If OVM = 0 (disabled counter is increment the counter is decre If OVM = 1 (enabled | d) and the operation genera- ted and if the operation gen mented. I), the counter is not affecte | ates a positive nerates a nega ed by the oper | e overfle ative ov ation. | ow, the rerflow, |
| | OVM | If overflow mode bi positive (0x7FFFFF overflowed. | t is set; then the ACC va F) or maximum negative (0 | lue will satur x80000000) if | ate ma the op | ximum eration |
| Repeat | | If this operation is re The state of the Z, N an intermediate ove overflows, if overflow | peated, then the instruction , C flags will reflect the final r erflow occurs. The OVC fl w mode is disabled. | will be execut esult. The V fla lag will count | ed N+1 ag will k : intern | times. be set if nediate |
| Example | ; Sub | tract three 32-bit | unsigned variables by | 16-bit part | s: | |
| | MOVU | ACC,@VarAlow | ; $AH = 0$, $AL = Var$ | Alow | | |
| | ADD | ACC,@VarAhigh << 1 | 6 ; AH = VarAhigh, A | AL = VarAlow | V | |
| | SUBU | ACC,@VarBlo293w | ; ACC = ACC - $0:Va$ | arBlow | | |
| | SUB | ACC,@VarBhigh << 1 | 6 ; ACC = ACC - VarB | 3high << 16 | | |
| | SBBU | ACC,@VarClow | ; $ACC = ACC - VarC$ | Clow - ~Carr | сy | |
| | SUB | ACC,@VarChigh << 1 | 6 ; ACC = ACC - VarC | Chigh << 16 | | |

SUBUL ACC, loc32

Subtract Unsigned 32-bit Value

| | SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|------------------|---|--|---|---------------------------------|-------------------------------------|
| SUBUL ACC, I | oc32 | | 0101 0110 0101 0101 0000 0000 LLLL LLLL | 1 | _ | 1 |
| Operands | loc32 | Addressing mode (s | see Chapter 5) | | | |
| | ACC | Accumulator registe | r | | | |
| Description | | Subtract from the A "loc32" addressing r operation: | ACC register the 32-bit the node. The subtraction is tre | location poir eated as an ur | nted to Isigned | by the SUBL |
| | | ACC = ACC - [loc | 32]; // unsigned sub | traction | | |
| | | Note: The difference b overflow count positive/negativ monitors the bo | etween a signed and unsigned 32- er (OVC). For a signed SUE e overflow. For an unsigned SUBL, rrow. | bit subtract is in th BL, the OVC c the OVC unsigne | e treatme counter d (OVCU | ent of the monitors) counter |
| Flags and Modes | Z | After the subtraction cleared. | n, the Z flag is set if the A | CC value is z | ero, el | se Z is |
| | Ν | After the subtraction cleared. | n, the N flag is set if bit 31 | of the ACC i | s 1, els | se N is |
| | С | If the subtraction ge | nerates a borrow, C is clea | red; otherwise | e C is s | et. |
| | V | If an overflow occur | s, V is set; otherwise V is n | ot affected. | | |
| | OVCU | The overflow count generates an unsign counter. | er is decremented whenev ned borrow. The OVM mode | ver a subtract e does not aff | tion op ect the | eration OVCU |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this in the repeat counter (RPTC) | struction follo and executes | ows the | e RPT once. |
| Example | ; Subt ; in V | ract two 64-bit va arC: | lues (VarA and VarB) a | nd store re | sult | |
| | MOVL | ACC,@VarA+0 | ; Load ACC with con : 32-bits of VarA | ntents of tl | ne low | |
| | SUBUL | ACC,@VarB+0 | ; Subtract from AC ; the low 32-bits | C the conter of VarB | nts of | |
| | MOVL | @VarC+0,ACC | ; Store low 32-bit | result into | o VarC | |
| | MOVL | ACC,@VarA+2 | ; Load ACC with co ; 32-bits of VarA | ntents of th | ne hig | h |
| | SUBBL | ACC,@VarB+2 | ; Subtract from AC : the high 32-bits | C the conter of VarB wit | nts of th bor | row |
| | MOVL | <pre>@VarC+2,ACC</pre> | ; Store high 32-bi | t result in | to Var | С |

SUBUL P, loc32

Subtract Unsigned 32-bit Value

| S | SYNTAX OPTIONS | | OPCODE | OBJMODE | RPT | CYC |
|--------------------|--|---|--|--|-----------------------------------|-----------------------------------|
| SUBUL P,loc32 | | | 0101 0110 0101 1101 0000 0000 LLLL LLLL | 1 | - | 1 |
| Operands | Р | Product register | | | | |
| - | loc32 | Addressing mode (s | ee Chapter 5) | | | |
| Description | | Subtract from the P the "loc32" addressi operation: | register the 32-bit content on ng mode. The addition is tra | of the location eated as an u | pointe nsigne | d to by d SUB |
| | | P = P - [loc32]; | <pre>// unsigned subtract</pre> | | | |
| | | Note: The difference be overflow counte positive/negative monitors the bor | etween a signed and unsigned 32-b er (OVC). For a signed SUB e overflow. For an unsigned SUBL, t row. | it subtract is in the L, the OVC c he OVC unsigned | e treatme ounter r d (OVCU) | ent of the monitors counter |
| Flags and Modes | Z | After the subtraction, | , the Z flag is set if the P valu | ue is zero, elso | e Z is c | leared. |
| | Ν | After the subtraction | , the N flag is set if bit 31 of | f P is 1, else I | l is cle | ared. |
| | С | If the subtraction ge | nerates a borrow, C is clear | ed; otherwise | C is s | et. |
| | V | If a signed overflow | occurs, V is set; otherwise | V is not affect | ed. | |
| | OVCU | The overflow counter generates an unsign counter. | er is decremented whenev ned borrow. The OVM mode | er a subtract does not affe | ion ope ect the | eration OVCU |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ws the only o | e RPT nce. |
| Example | ; Subtr MOVL F MOVL A SUBUL F | ract 64-bit VarA - P,@VarA+0 ; Ld ACC,@VarA+2 ; Ld P,@VarB+0 ; St | VarB and store result oad P with low 32-bits oad ACC with high 32-b ub from P unsigned low | in VarC: of VarA its of VarA 32-bits of | VarB | |
| | SUBBL A | VarC+0,P ; S | ub from ACC with borrow tore low 32-bit result | w nigh 32-b: into VarC | its of | VarB |

MOVL @VarC+2,ACC ; Store high 32-bit result into VarC

TBIT loc16,#bit

Test Specified Bit

| S | SYNTAX OPT | IONS | OPCODE | OBJMODE | RPT | CYC |
|--------------------|---------------------------------------|---|---|---|---------------------------------|--------------------------------|
| TBIT loc16,#16 | 6bit | | 0100 BBBB LLLL LLLL | Х | - | 1 |
| Operands | loc16#bit | Addressing mode Immediate consta | e (see Chapter 5) ant bit index from 0 to 15 | | | |
| Description | | Test the specified "loc16" addressin | l bit of the data value in the g mode: | e location poir | nted to | by the |
| | | TC = [loc16(k | pit)]; | | | |
| | | The value specifie to the bit number significant bit) of t (most significant l | ed for the #bit immediate op . For example, if #bit = 0, he addressed location; if #bi bit). | perand directly you will acce t = 15, you wil | / corres ss bit (l acces | sponds) (least s bit 15 |
| Flags and Modes | тс | If the bit tested is | 1, TC is set; if the bit tested | d is 0, TC is c | leared. | |
| Repeat | | This instruction i instruction, it rese | s not repeatable. If this ir ets the repeat counter (RPT) | nstruction follo C) and execut | ows th tes only | e RPT / once. |
| Example | ; if(Var ; Var ; else ; Var | A.Bit4 = 1) B.Bit6 = 1; B.Bit6 = 0; | | | | |
| | TBIT | @VarA,#4 | ; Test bit 4 of VarA of | contents | | |
| | SB | SIO,NTC | ; Branch if $TC = 0$; Set bit 6 of VarP of | ontente | | |
| | SB | \$20.UNC | : Branch unconditional | llv | | |
| | \$10: | , | ; | 4 | | |
| | TCLR | @VarB,#6 | ; Clear bit 6 of VarB | contents | | |
| | \$20: | | ; | | | |

| TBIT loc16,T Test Bit Specified by Register | | | | | | |
|---|---|--|--|--|----------------------|---------------------|
| | SYNTAX OF | PTIONS | OPCODE | OBJMODE | RPT | CYC |
| TBIT loc16,T | | | 0101 0110 0010 0101 0000 0000 LLLL LLLL | 1 | - | 1 |
| Operands | loc16 T | Addressing mode (Upper 16 bits of the | (see Chapter 5) e multiplicand register (XT) | | | |
| Description | | Test the bit specifi T(3:0) = 015 of t addressing mode. | ied by the four least signif he data value in the location Upper bits of the T register | icant bits of t on pointed to are ignored: | he T ro by the | egister, "loc16" |
| | | bit = 15 - T(3:0 TC = [loc16(bit) | 0);)]; | | | |
| | | A value of 15 in the value of 0 in the T r upper 12 bits of the | e T register corresponds to l register corresponds to bit 1 e T register are ignored. | bit 0 (least sig 5 (most signifi | nificant icant bi | bit). A t). The |
| Flags and Modes | тс | If the bit tested is 1 | , TC is set; if the bit tested | is 0, TC is cle | ared. | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ir sthe repeat counter (RPTC | nstruction follo () and execute | ows th s only | e RPT once. |
| Example | ; if(V; ; V; ; else ; V; MOV ADD TBIT SB TSET SB \$10: | arA.VarB = 1) arC.Bit6 = 1; arC.Bit6 = 0; T,@VarB @T,#15 @VarA,T \$10,NTC @VarB,#6 \$20,UNC | ; Load T with bit val ; Reverse order of bi ; Test bit of VarA se ; Branch if TC = 0 ; Set bit 6 of VarB c ; Branch unconditiona ; | ue in VarB t testing lected by V ontents lly | arB | |
| | TCLR | @VarB,#6 | , ; Clear bit 6 of VarB | contents | | |
| | \$20: | | i | | | |

TCLR loc16,#bit

Test and Clear Specified Bit

| | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------|--|--|--|--------------------------------------|--------------|-------|--|--|
| TCLR loc16,# | bit | 0101 0110 0000 1001 1 0000 BBBB LLLL LLLL | | | | | | |
| Operands | loc16, #bit | Addressing mode (s | see Chapter 5) | | | | | |
| | | Immediate constant | bit index from 0 to 15 | | | | | |
| Description | | Test the specified bit of the data value in the location pointed to by th addressing mode and then clear that same bit to 0: | | | | | | |
| | | TC = [loc16(bit) [loc16(bit)] = 0 | TC = [loc16(bit)]; [loc16(bit)] = 0; | | | | | |
| | | The value specified for the #bit immediate operand directly corresponds the bit number. For example, if #bit = 0, you will access bit 0 (least significa bit) of the addressed location; if #bit = 15, you will access bit 15 (mo significant bit). TCLR performs a read-modify-write operation. | | | | | | |
| Flags and | Ν | If (loc16 = $@AX$) and bit 15 (MSB) of $@AX$ is 1, then N flag is set | | | | | | |
| Modes | Z TC | If (loc16 = @AX) an If the bit tested is 1, | d @AX gets zeroed out, t TC is set; if the bit tested | hen Z flag is se is 0, TC is clea | et. ared. | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this the repeat counter (RPT) | nstruction follo C) and executes | ows the | e RPT | | |
| Example | ; if(); ; '' ; else ; '' TBII | VarA.Bit4 = 1) VarB.Bit6 = 1; VarB.Bit6 = 0; T @VarA,#4 ; Test bit 4 of VarA contents | | | | | | |
| | SB TSE: | $\Gamma = @VarB, #6$ | ; Branch II IC = ; Set bit 6 of V | ∶ ∪ /arB contents | 5 | | | |
| | SB | \$20,UNC | ; Branch uncond | tionally | | | | |
| | \$10: | | ; | | | | | |
| | TCLI | R @VarB,#6 | ; Clear bit 6 of | VarB conter | its | | | |
| | \$20: | | ; | | | | | |

TEST ACC

Test for Accumulator Equal to Zero

| S | SYN | ΙΤΑΧ Ο | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|--------------------|-----|--------------------|---|---|--------------|-------|-----|--|
| TEST ACC | | | | 1111 1111 0101 1000 | Х | - | 1 | |
| Operands | Α | сс | Accumulator register | | | | | |
| Description | | | Compare the ACC register to zero and set the status flag bits accordingl Modify flags on (ACC - 0x00000000); | | | | | |
| Flags and Modes | Ν | | If bit 31 of the ACC is 1, N is set; else N is cleared. | | | | | |
| | z | | If ACC is zero, Z is set; else Z is cleared. | | | | | |
| Repeat | | | This instruction is not repeatable. If this instruction follows the RPT in- struction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; | Test TEST SB | contents of ACC an ACC Zero,EQ | nd branch if zero: ; Modify flags or ; Branch if zero | n (ACC - 0x(| 00000 | 00) | |

| TRAP #VectorNumber Software Trap | | | | | | e Trap |
|--|------------------|---|---|--|--|--|
| | SYNTAX OP | TIONS | OPCODE | OBJMODE | RPT | CYC |
| TRAP #VectorNumber | | | 0000 0000 001C CCCC | Х | - | 8 |
| Operands | Vector Number | CPU interrupt vect | tor 0 to 31 | | | |
| Description | | The TRAP instruct routine that corres not affect the inter (IER), regardless of these registers. The mask bit (INTM) in in the IER or the de- instruction reaches cannot be service of the following table chosen value for the | tion transfers program of ponds to the vector speci rupt flag register (IFR) of f whether the chosen inter e TRAP instruction is not a status register ST1. It also abug interrupt enable regis is the decode phase of the d until the TRAP instruction butine begins). e indicates which interrupt the VectorNumber operation | ontrol to the int fied in the instru- r the interrupt e rrupt has corres affected by the i not affected by ter (DBGIER). C pipeline, hardv on is done exect ot vector is asse | errupt uction. nable r ponding nterrup the ena Droce the vare int uting (u | service It does register g bits in t global uble bits e TRAP errupts until the with a |
| Vector | Interrupt | Vector | | nterrupt | | |
| Number | Vector | Number | N | Vector ['] | | |
| 0 | RESET | 16 | I | RTOSINT | | |
| 1 | INT1 | 17 | I | Reserved | | |
| 2 | INT2 | 18 | I | IMV | | |
| 3 | INT3 | 19 | I | LLEGAL | | |
| 4 | INT4 | 20 | l | JSER1 | | |
| 5 | INT5 | 21 | l | JSER2 | | |
| 6 | INT6 | 22 | l | JSER3 | | |
| 7 | INT7 | 23 | l | JSER4 | | |
| 8 | INT8 | 24 | l | JSER5 | | |
| 9 | INT9 | 25 | l | JSER6 | | |
| 10 | INT10 | 26 | l | JSER7 | | |
| 11 | INT11 | 27 | l | JSER8 | | |
| 12 | INT12 | 28 | l | JSER9 | | |
| 13 | INT13 | 29 | l | JSER10 | | |
| 14 | INT14 | 30 | l | JSER11 | | |
| 15 | DLOGINT | 31 | l | JSER12 | | |

Part of the operation involves saving pairs of 16-bit core registers onto the stack pointed to by the SP register. Each pair of registers is saved in a single 32-bit operation. The register forming the low word of the pair is saved first (to an even address); the register forming the high word of the pair is saved next (to the following odd address). For example, the first value saved is the concatenation of the T register and the status register ST0 (T:ST0). ST0 is saved first, then T.

This instruction should not be used with vectors 1-12 when the peripheral interrupt expansion (PIE) is enabled.

Note: The TRAP #0 instruction does not initiate a full reset. It only forces execution of the interrupt service routine that corresponds to the RESET interrupt vector.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

TSET loc16,#16bit Test and Set Specified Bit SYNTAX OPTIONS OBJMODE RPT OPCODE CYC TSET loc16,#16bit 0101 0110 0000 1101 1 _ 1 0000 BBBB LLLL LLLL Operands loc16 Addressing mode (see Chapter 5) #bit Immediate constant bit index from 0 to 15 Description Test the specified bit of the data value in the location pointed to by the "loc16" addressing mode and then set the same bit to 1: TC = [loc16(bit)];[loc16(bit)] = 1;The value specified for the #bit immediate operand directly corresponds to the bit number. For example, if #bit = 0, you will access bit 0 (least significant bit) of the addressed location; if #bit = 15, you will access bit 15 (most significant bit). TSET performs a read-modify-write operation. Flags and Ν If (loc16 = @AX) and bit 15 (MSB) of @AX is 1, then N flag is set. Modes Ζ If (loc16 = @AX) and @AX gets zeroed out, then Z flag is set. тс If the bit tested is 1, TC is set; if the bit tested is 0, TC is cleared. Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. ; if(VarA.Bit4 = 1) Example VarB.Bit6 = 1;; else ; VarB.Bit6 = 0;; TBIT @VarA,#4 ; Test bit 4 of VarA contents ; Branch if TC = 0SB \$10,NTC @VarB,#6 ; Set bit 6 of VarB contents TSET SB \$20,UNC ; Branch unconditionally \$10: ; TCLR @VarB,#6 ; Clear bit 6 of VarB contents \$20: ;

UOUT *(PA),loc16

Unprotected Output Data to I/O Port

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|----------------|--|---|----------------------------------|--------------------|-------------------|--|--|
| UOUT *(PA),lo | c16 | | 1011 0000 LLLL LLLL CCCC CCCC CCCC CCCC | 1 | Y | N+2 | | |
| Operands | *(PA) loc16 | Immediate I/O space memory address | | | | | | |
| | | Addressing mode (see Chapter 5) | | | | | | |
| Description | | Store the 16-bit valu mode into the I/O sp | 6-bit value from the location pointed to by the "loc16" addressing the I/O space location pointed to by "*(PA): | | | | | |
| | | IOspace[0x000:PA |] = loc16; | | | | | |
| | | I/O Space is limited to 64K range (0x0000 to 0xFFFF). Or interface (XINTF), if available on a particular device, the I/O (XISn) is toggled during the operation. The I/O address appear 16 address lines (XA(15:0)) and the upper address lines are data appears on the lower 16 data lines (XD(15:0). | | | | | | |
| | | Note: The UOUT operation is not pipeline protected. Therefore, if immediately follows a UOUT instruction, the IN will occur before certain of the sequence of operation, use the OUT instruction protected. I/O space may not be implemented on all C28x devices. See the particular device for details. | | | | | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is re it will be executed N- is post-incremented | peatable. If the operation foll +1 times. When repeated, th by 1 during each repetition | ows a RPT in: e "*(PA)" I/O s | structio pace a | n, then ddress | | |

```
Example
            ; IORegA address = 0x0300;
            ; IOREgB address = 0x0301;
            ; IOREgC address = 0x0302;
            ; IORegA = 0x0000;
            ; IORegB = 0x0400;
            ; IORegC = VarA;
            ; if ( IORegC = 0x2000 )
            ; IORegC = 0x0000;
            IORegA .set 0x0300 ; Define IORegA address
IORegB .set 0x0301 ; Define IORegB address
            IORegC .set 0x0302 ; Define IORegC address
                                    ; AL = 0
              MOV
                    @AL,#0
                    *(IORegA),@AL ; IOspace[IORegA] = AL
              UOUT
              MOV
                     @AL, #0x0400; AL = 0x0400
              UOUT
                     *(IORegB),@AL ; IOspace[IORegB] = AL
              OUT
                     *(IO-
                                    ; IOspace[IORegC] = VarA
            RegC),@VarA
              IN
                     @AL,*(IORegC) ; AL = IOspace[IORegC]
              CMP
                     @AL,#0x2000 ; Set flags on (AL - 0x2000)
                     $10,NEQ
                                   ; Branch if not equal
              SB
                                    ; AL = 0
              MOV
                     @AL,#0
              UOUT
                    *(IORegC),@AL ; IOspace[IORegC] = AL
            $10:
```

| XB *AL | | | | C2 xLP Source-(| Compatible Ind | direct B | ranch |
|--------------------|---|--|------------------------------------|---|--|---------------------|---------------|
| 9 | Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο Ο | DNS | C | PCODE | OBJMODE | RPT | CYC |
| XB *AL | | | 0101 011 | 0 0001 0100 | 1 | - | 7 |
| Operands | *AL Inc hig | lirect program-me h 64K of program | mory add space rai | ressing using re nge (0x3F0000 t | gister AL, car o 0x3FFFFF) | n only a | access |
| Description | Un coi | conditional indired | ct branch | by loading the cing the upper 6 | low 16 bits of bits of the PC | FPC w to 0x3 | ith the F: |
| | PC | PC = 0x3F:AL; | | | | | |
| | Not | te: This branch instru program space ((| uction can or 0x3F0000 to | ly branch to a locatio 0x3FFFFF). | n located in the up | oper 64K | range of |
| Flags and Modes | No | ne | | | | | |
| Repeat | Th ins | is instruction is truction, it resets t | not repea he repeat | table. If this in counter (RPTC) | struction follo and executes | ows the s only o | ∍ RPT nce. |
| Example | ; Branch t ; value. ; This exa ; program SwitchTabl .word .word | o subroutines i mple only works space: e: Switch0 Switch1 | n Switch for cod ; ; ; | Table selecte de located in Switch addres Switch0 addre Switch1 addre | d by Switch upper 64K o s table: ss ss | of | |
| | MOVL MOVZ MOV XB SwitchRetu | XAR2,#SwitchTa AR0,@Switch AL,*+XAR2[AR0] *AL rn: | uble ; ; ; ; | XAR2 = pointe AR0 = Switch AL = SwitchTa Indirect bran | r to Switch index ble[Switch] ch using AL | Table | |
| | Switch0: | | ; | Subroutine 0: | | | |
| | XB | SwitchReturn,U | INC ; | Return: branc | h | | |
| | Switchl: XB | SwitchReturn | ; INC · | Subroutine 1: | h | | |
| | | owiccinceulli, c | //// | Mecurii, Draile | 11 | | |

XB pma,*,ARPn

C2xLP Source-Compatible Branch with ARP Modification

| S | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|-------------------------------------|---|--|---------------------|----------|----------|--|--|
| XB pma,*,ARP | 'n | | 0011 1110 0111 0nnn CCCC CCCC CCCC CCCC | 1 | - | 4 | | |
| Operands | pma | 16-bit immediate pro | ogram -memory address, 164K of program space rand | ae (0x3F0000 | to 0x3F | FFFF) | | |
| | ARPn | 3-bit auxiliary registe | bit auxiliary register pointer (ARP0 to ARP7) | | | | | |
| Description | | Unconditional branc with the 16-bit immer to 0x3F. Also, chan "ARPn" operand: | Jnconditional branch with ARP modification by loading the low 16 bits of PC with the 16-bit immediate value "pma" and forcing the upper 6 bits of the PC o 0x3F. Also, change the auxiliary register pointer as specified by the ARPn" operand: | | | | | |
| | | PC = 0x3F:pma; ARP = n; | PC = 0x3F:pma; ARP = n; | | | | | |
| | | Note: This branch instr program space (| uction can only branch to a locatior (0x3F0000 to 0x3FFFFF). | n located in the up | oper 64K | range of | | |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | |
| Example | ; Branc ; by AR ; locat XB | n to SubA and set ARP. Load ACC with pointer pointed to P and return to. This example only works for code ed in upper 64K of program space: SubA,*,ARP1 ; Branch to SubA with ARP pointing | | | | | | |
| | SubRetu | rn: | ; to XAR1 | | | | | |
| | | | | | | | | |
| | SubA: MOV | L ACC,* | ; Subroutine A: ; Load ACC with | contents | | | | |
| | XB | SubReturn, UNC | ; poinced to by ; ; Return uncondi | tionally | | | | |

XB pma,COND

C2 xLP Source-Compatible Branch

| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|------|---------------------------------------|---|---|---|---------------------------------|------------------------------|--|
| XB pma,COND | | | | 0101 0110 1101 COND CCCC CCCC CCCC CCCC | 1 | _ | 7/4 | |
| Operands | pma | 16-bit i prograr | 16-bit immediate program-memory address, can only access high 64 program space range (0x3F0000 to 0x3FFFFF) | | | | | |
| | COND | Condit | ional code | s: | | | | |
| | | COND | Syntax | Description | Flags | Teste | d | |
| | | 0000 | NEQ | Not Equal To | Z = 0 | | | |
| | | 0001 | EQ | Equal To | Z = 1 | | | |
| | | 0010 | GT | Greater Then | Z = 0 | AND N | = 0 | |
| | | 0011 | GEQ | Greater Then Or Equal | To $N = 0$ | | | |
| | | 0100 | LT | Less Then | N = 1 | | | |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 | OR N | = 1 | |
| | | 0110 | HI | Higher | C = 1 | AND Z | = 0 | |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 | | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | | |
| | | 1001 | LOS | Lower Or Same | C = 0 | OR Z | = 1 | |
| | | 1010 | NOV | No Overflow | V = 0 | | | |
| | | 1011 | VO | Overflow | V = 1 | | | |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 |) | | |
| | | 1101 | ТС | Test Bit Set | TC = 2 | L | | |
| | | 1110 | NBIO | BIO Input Equal To Zer | CO BIO = | 0 | | |
| | | 1111 | UNC | Unconditional | - | | | |
| Description | | Conditi the low upper branch | onal branch. 16 bits of PC 6 bits of the ing: | If the specified condition is tr C with the 16-bit immediate v PC to 0x3F.; otherwise c | rue, then brar ralue "pma" a ontinue exec | nch by l nd forc cution N | oading ing the without | |
| | | If (CC If (CC PC(21: | PND = true) PND = false 16) = 0x3F | PC(15:0) = pma;) PC(15:0) = PC(15:0) +; | 2; | | | |
| | | Note: | If (COND = true) If (COND = false |) then the instruction takes 7 cycles e) then the instruction takes 4 cycle | s. S. | | | |
| Flags and Modes | V | If the V | flag is tested | d by the condition, then V is | cleared. | | | |
| Repeat | | This in instruct | struction is ion, it resets | not repeatable. If this ins | struction follo | ows the s only c | e RPT ince. | |

```
Example
          ; Branch to subroutines in SwitchTable selected by Switch value.
          ; This example only works for code located in upper 64K of
          ; program space:
          SwitchTable:
                                      ; Switch address table:
             .word Switch0
.word Switch1
                                     ; Switch0 address
                                     ; Switch1 address
               .
              •
             MOVL XAR2, #SwitchTable ; XAR2 = pointer to SwitchTable
             *AL
                                     ; Indirect branch using AL
             XB
          SwitchReturn:
               .
          Switch0:
                                     ; Subroutine 0:
              •
               .
             XB
                  SwitchReturn,UNC
                                    ; Return: branch
          Switch1:
                                      ; Subroutine 1:
              .
             XB SwitchReturn, UNC ; Return: branch
```

| XBANZ pma,*ind{ | ARPn} | C2 x LP Source-Com | npatible Branch | n If ARn Is Not Zero |
|-----------------|-------|--------------------|-----------------|----------------------|
|-----------------|-------|--------------------|-----------------|----------------------|

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|---------------------|--|---------|-----|-----|
| XBANZ pma,* | 0101 0110 0000 1100 CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*++ | 0101 0110 0000 1010 CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,* | 0101 0110 0000 1011 CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*0++ | 0101 0110 0000 1110 CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*0 | 0101 0110 0000 1111 CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*,ARPn | 0011 1110 0011 0nnn CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*++,ARPn | 0011 1110 0011 1nnn CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*,ARPn | 0011 1110 0100 0nnn CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*0++,ARPn | 0011 1110 0100 1nnn CCCC CCCC CCCC CCCC | 1 | - | 4/2 |
| XBANZ pma,*0,ARPn | 0011 1110 0101 0nnn CCCC CCCC CCCC CCCC | 1 | - | 4/2 |

Operands pma

16-bit immediate program-memory address, can only access high 64K of program space range (0x3F0000 to 0x3FFFFF)

ARPn 3-bit auxiliary register pointer (ARP0 to ARP7)

Description If the lower 16 bits of the auxiliary register pointed to by the current auxiliary register pointer (ARP) is not equal to 0, then a branch is taken by loading the lower 16 bits of the PC with the 16-bit immediate "pma" value and forcing the upper 6 bits of the PC to 0x3F. Then, the current auxiliary register, pointed to by the ARP, is modified as specified by the indirect mode. Then, if indicated, the ARP pointer value is changed to point a new auxiliary register:

```
if( AR[ARP] != 0 )
PC = 0x3F:pma
if(*++ indirect mode) XAR[ARP] = XAR[ARP] + 1;
if(*-- indirect mode) XAR[ARP] = XAR[ARP] - 1;
if(*0++ indirect mode) XAR[ARP] = XAR[ARP] + AR0;
if(*0-- indirect mode) XAR[ARP] = XAR[ARP] - AR0;
if(ARPn specified) ARPn = n;
```

Note: This instruction can only transfer program control to a location located in the upper 64K range of program space (0x3F0000 to 0x3FFFFF). The cycle times for this operation are:

If branch is taken, then the instruction takes 4 cycles

If branch is not taken, then the instruction takes 2 cycles

| Flags and Modes | I | None | | |
|--------------------|---|--|---|--|
| Repeat | - i | This instruction is nstruction, it resets t | not he r | repeatable. If this instruction follows the RPT repeat counter (RPTC) and executes only once. |
| Example | ; Copy t ; int32 ; int32 ; for(i= ; Arra ; This e ; progra MOVL MOVL MOV NOP SETC Loop: MOVL MOVL BANZ | he contents of Ar Array1[N]; Array2[N]; 0; i < N; i++) y2[i] = Array1[i] xample only works m space: XAR2,#Array1 XAR3,#Array2 @AR0,#(N-1) *,ARP2 AMODE ACC,*++,ARP3 *++,ACC,ARP0 Loop,*,ARP2 | ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; | <pre>rl to Array2: pr code located in upper 64K of XAR2 = pointer to Array1 XAR3 = pointer to Array2 Repeat loop N times Point to XAR2 Full C2XLP address mode compatible ACC = Array1[i], point to XAR3 Array2[i] = ACC, point to XAR0 Loop if AR[ARP] != 0, AR[ARP], point to XAR2</pre> |

| XCALL *AL | C2 x LP Source-Compatible Function Call | | | | | | | | |
|--------------------|---|--|---------------------------------|---|---|--------------------------------|-------------------------------|--|--|
| S | YNTAX OPT | ONS | | OPCODE | OBJMODE | RPT | CYC | | |
| XCALL *AL | | | 0101 | 0110 0011 0100 | 1 | - | 7 | | |
| Operands | * AL In hi | *AL Indirect program-memory addressing using register AL, can only access high 64K of program space range (0x3F0000 to 0x3FFFFF) | | | | | | | |
| Description | In P ⁱ Io Io | direct call with des C address are save aded with the cont aded with 0x3F: | tination d onto tents o | n address in AL. The the software stack. T f register AL and the | lower 16 bits hen, the low 1 upper 6 bits | of the 6 bits o of the l | current of PC is PC are | | |
| | te [1 S1 C | <pre>temp(21:0) = PC + 1; [SP] = temp(15:0); SP = SP + 1; C = 0x3F:AL;</pre> | | | | | | | |
| | N | ote: I his instruction of range of program XCALL, the XRE | can only m space TC insti | transfer program control to (0x3F0000 to 0x3FFFFF ruction must be used. | a location located -). To return from | l in the u n a call i | oper 64K nade by | | |
| Flags and Modes | Ν | one | | | | | | | |
| Repeat | Ti in | his instruction is struction, it resets | not re the rep | epeatable. If this inspectation to the termination of | struction follo and executes | ows the s only c | e RPT | | |
| Example | ; Call fu ; This ex ; program | nction in FuncTa ample only works space: | able s s for | selected by FuncIn code located in | ndex value. upper 64K d | of | | | |
| | FuncTable | : | ; | Function address | table: | | | | |
| | .word | FuncA | ; | FuncA address | | | | | |
| | .word | FuncB | ; | FuncB address | | | | | |
| | • | | | | | | | | |
| | MOVL | XAR2, #FuncTab | le ; | XAR2 = pointer to | o FuncTable | | | | |
| | MOVZ | AR0,@FuncIndez | x; | AR0 = FuncTable | index | | | | |
| | MOV | AL,*+XAR2[AR0] |] ; | AL = Table[FuncI | ndex] | | | | |
| | XCALL | *AL | ; | Indirect call us | ing AL | | | | |
| | • | | | | | | | | |
| | FuncA: | | ; | Function A: | | | | | |
| | XRETC | UNC | ; | Return unconditio | onally | | | | |
| | FuncB: | | ; | Function B: | | | | | |
| | XRETC | UNC | ; | Return unconditio | onally | | | | |

| XCALL pma | ,*,ARPn | | C2 x LP Source-Compatible Function Call | | | | |
|--------------------|-------------------------------|---|--|---|--------------------------|---------------------|--|
| S | SYNTAX OF | PTIONS | OPCODE OBJMODE RPT CYC | | | | |
| XCALL pma,*, | ARPn | | 0011 1110 0110 1nnn CCCC CCCC CCCC CCCC | 1 | - | 4 | |
| Operands | pma | 16-bit immediate pro | ogram-memory address, 64K of program space ra | unae (0x3F0000 | to 0x3F | FFFF) | |
| | ARPn | 3-bit auxiliary registe | er pointer (ARP0 to ARP | 7) | | , | |
| Description | | Unconditional call with ARP modification. The lower 16 bits of the retu address are pushed onto the software stack. Then, the lower 16 bits of the PC are loaded with the 16-bit immediate "pma" value and the upper 6 bits the PC are forced to 0x3F. Then, the 3-bit ARP pointer will be set to the "ARPn" field value: | | | | | |
| | | temp(21:0) = PC [SP] = temp(15:0) SP = SP + 1; PC = 0x3F:pma; ARP = n; | + 1;); | | | | |
| | | Note: This instruction of range of progra XCALL, the XRE | can only transfer program contro m space (0x3F0000 to 0x3FFI ETC instruction must be used. | to a location located FFF). To return fror | d in the u n a call i | pper 64K made by | |
| Flags and Modes | | None | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this the repeat counter (RPT | instruction follo | ows the | e RPT once. | |
| Example | ; Call ; ; This ; space | FuncA and set ARP example only work : LL FuncA * ARP1 | . Load ACC with point s for code located in | ter pointed t n upper 64K o | o by A f prog | RP. Tram | |
| | • | III FUICA, , ARTI | , call funck with | ARI poincing | | ART | |
| | FuncA: MOV | L ACC,* | ; Function A: ; Load ACC with c | ontents point | ed to | | |
| | XRE' | TC UNC | ; Dy JAR(ARP) ; Return uncondit | ionally | | | |

XCALL pma,COND

C2xLP Source-Compatible Function Call

| | SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
|--|----------------|--------------------|--------------------------------|--|--------------|--|-----------------------------------|
| XCALL pma,C | OND | | | 0101 0110 1110 COND CCCC CCCC CCCC CCCC | 1 | - | 7/4 |
| Operands | pma | 16-bit i can on | mmediate pro ly access high | ogram-memory address, 64K of program space rang | ge (0x3F0000 | to 0x3 | FFFF) |
| | COND | Conditi | ional codes: | | | | , |
| | | COND | Syntax | Description | Flags | Teste | d |
| | | 0000 | NEQ | Not Equal To | Z = 0 | | |
| | | 0001 | EQ | Equal To | Z = 1 | | |
| | | 0010 | GT | Greater Then | Z = 0 | AND N | = 0 |
| | | 0011 | GEQ | Greater Then Or Eq To | ual N = O | | |
| | | 0100 | LT | Less Then | N = 1 | | |
| | | 0101 | LEQ | Less Then Or Equal | . To Z = 1 | OR N | = 1 |
| | | 0110 | HI | Higher | C = 1 | AND Z | = 0 |
| | | 0111 | HIS, C | Higher Or Same, Ca Set | erry C = 1 | | |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 | | |
| | | 1001 | LOS | Lower Or Same | C = 0 | OR Z | = 1 |
| | | 1010 | NOV | No Overflow | V = 0 | | |
| | | 1011 | OV | Overflow | V = 1 | | |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 | C | |
| | | 1101 | TC | Test Bit Set | TC = 1 | 1 | |
| | | 1110 | NBIO | BIO Input Equal To Zero | BIO = | 0 | |
| | | 1111 | UNC | Unconditional | _ | | |
| Description Conditional call. If the specified condition is true, then the low 16 bits return address is pushed onto the software stack and the low 16 bits PC are loaded with the 16-bit immediate "pma" value and the upper 6 the PC are forced to 0x3F; otherwise continue execution with instribution of the XCALL operation: | | | | | | s of the s of the 5 bits of ruction | |
| <pre>{ temp(21:0) = PC + 2; [SP] = temp(15:0); SP = SP + 1; PC = 0x3F:pma; } else PC = PC + 2;</pre> | | | | | | | |
| Note: This instruction can only transfer program control to a location located in the upp range of program space (0x3F0000 to 0x3FFFFF). To return from a call ma XCALL, the XRETC instruction must be used. The cycle times for this operation If (COND = true) then the instruction takes 7 cycles. If (COND = false) then the instruction takes 4 cycles. | | | | | | | oper 64K made by ation are: |

| Flags and Modes | V | If the V flag | If the V flag is tested by the condition, then V is cleared. | | | | |
|--------------------|-----------------------------------|--|--|---|--|--|--|
| Repeat | | This instruc instruction, i | tion is not re t resets the re | repeatable. If this instruction follows the RPT epeat counter (RPTC) and executes only once. | | | |
| Example | ; Cal ; wor M X FuncA | l FuncA if Va ks for code l OV AL,@Var CALL FuncA,N | urA does not ocated in u TA ; TEQ ; ; | t equal zero. This example only upper 64K of program space: ; Load AL with VarA ; Call FuncA if not equal to zero ; Function A: | | | |
| | Х | RETC UNC | ; | ; Return unconditionally | | | |

XMAC P,loc16,*(pma)

C2xLP Source-compatible Multiply and Accumulate

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|---------------------|--|---------|-----|-----|
| XMAC P,loc16,*(pma) | 1000 0100 LLLL LLLL CCCC CCCC CCCC CCCC | 1 | Y | N+2 |

| Operands | Ρ | Product register | | | | | |
|--------------------|--------|--|--|--|--|--|--|
| | loc16 | Addressing mode (see Chapter 5) | | | | | |
| | *(pma) | Immediate program memory address, access high 64K range of program space only (0x3F0000 to 0x3FFFFF) | | | | | |
| Description | | Add the previous product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. Next, load the T register with the content of the location pointed to by the "loc16" addressing mode. Last, multiply the signed 16-bit content of the T register by the signed 16-bit content of the addressed program memory location and store the 32-bit result in the P register: | | | | | |
| | | ACC = ACC + P << PM; T = [loc16]; P = signed T * signed Prog[0x3F:pma]; | | | | | |
| | | The C28x forces the upper 6 bits of the program memory address, specified by the "*(pma)" addressing mode, to 0x3F when using this form of the MAC instruction. This limits the program memory address to the high 64K of program address space (0x3F0000 to 0x3FFFFF). On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*(pma)" addressing mode can be used to access data space variables that fall within its address range. | | | | | |
| Flags and Modes | Z | After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. | | | | | |
| | Ν | After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared. | | | | | |
| | С | If the addition generates a carry, C is set; otherwise C is cleared. | | | | | |
| | V | If an overflow occurs, V is set; otherwise V is not affected. | | | | | |
| | ovc | If overflow mode is disabled; and if the operation generates a positive overflow, then the counter is incremented. If overflow mode is disabled; and if the operation generates a negative overflow, then the counter is decremented. | | | | | |
| | OVM | If overflow mode bit is set; then the ACC value will saturate maximum positive (0x7FFFFFF) or maximum negative (0x80000000) if the operation overflowed. | | | | | |

| | РМ | The value in the PM k product register. If t operation), then the negative (arithmetic r | oits : he low ight | sets the shift mode for the output operation from the product shift value is positive (logical left shift bits are zero filled. If the product shift value is t shift operation), the upper bits are sign extended. | | |
|---------|--|---|-----------------------------|--|--|--|
| Repeat | | This instruction is rep then it will be execute will reflect the final re occurs. When repeat 1 during each repetit | ed N suli ed, ion. | able. If the operation follows a RPT instruction, N+1 times. The state of the Z, N, C and OVC flags t. The V flag will be set if an intermediate overflow the program-memory address is incremented by | | |
| Example | ; Calcul ; int16 ; int16 ; sum = ; for(i= ; sum | <pre>late sum of product using 16-bit multiply: X[N] ; Data information C[N] ; Coefficient information, located in high 64K 0; =0; i < N; i++)</pre> | | | | |
| | MOVL | XAR2,#X | ; | XAR2 = pointer to X | | |
| | SPM | -5 | ; | Set product shift to ">> 5" | | |
| | ZAPA | | ; | Zero ACC, P, OVC | | |
| | RPT | #N-1 | ; | Repeat next instruction N times | | |
| | XMAC | P,*XAR2++,*(C) | ; ; | ACC = ACC + P >> 5, P = *XAR2++ * *C++ | | |
| | ADDL | ACC,P << PM | ; | Perform final accumulate | | |
| | MOVL | @sum,ACC | ; | Store final result into sum | | |

XMACD P,loc16,*(pma) C2xLP Source-Compatible Multiply and Accumulate With Data Move

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------------|--|---------|-----|-----|
| XMACD P,loc16,*(pma) | 1010 0100 LLLL LLLL CCCC CCCC CCCC CCCC | 1 | Y | N+2 |

Operands P Product register

- Ioc16 Addressing mode (see Chapter 5) Note: For this operation, register-addressing modes cannot be used. The modes are: @ARn, @AH, @AL, @PH, @PL, @SP, @T. An illegal instruction trap will be generated.
- *(pma) Immediate program memory address, access high 64K range of program space only (0x3F0000 to 0x3FFFFF)
- **Description** The XMACD instruction functions in the same manner as the XMAC, with the addition of a data move. Add the previous product (stored in the P register), shifted as specified by the product shift mode (PM), to the ACC register. Next, load the T register with the content of the location pointed to by the "loc16" addressing mode. Then, multiply the signed 16-bit content of the T register by the signed 16-bit content of the addressed program memory location and store the 32-bit result in the P register. Last, store the content in the T register onto the next highest memory address pointed to by "loc16" addressing mode:

ACC = ACC + P << PM; T = [loc16]; P = signed T * signed Prog[0x3F:pma]; [loc16 + 1] = T;

The C28x forces the upper 6 bits of the program memory address, specified by the "*(pma)" addressing mode, to 0x3F when using this form of the MAC instruction. This limits the program memory address to the high 64K of program address space (0x3F0000 to 0x3FFFFF). On the C28x devices, memory blocks are mapped to both program and data space (unified memory), therefore, the "(pma)" addressing mode can be used to access data-space variables that fall within its address range.

- **Flags and Z** After the addition, the Z flag is set if the ACC value is zero, else Z is cleared. **Modes**
 - **N** After the addition, the N flag is set if bit 31 of the ACC is 1, else N is cleared.
 - **C** If the addition generates a carry, C is set; otherwise C is cleared.
 - V If an overflow occurs, V is set; otherwise V is not affected.
 - **OVC** If overflow mode is disabled and if the operation generates a positive overflow, the counter is incremented. If overflow mode is disabled and if the operation generates a negative overflow, the counter is decremented.

| | OVM | If overflow mode bit is (0x7FFFFFFF) or m overflowed. | s set, the ACC value will saturate maximum positive aximum negative (0x80000000) if the operation |
|---------|---|---|--|
| | РМ | The value in the PM bi product register. If the operation), then the negative (arithmetic ri | its sets the shift mode for the output operation from the ne product shift value is positive (logical left shift ow bits are zero filled. If the product shift value is ght shift operation), the upper bits are sign extended. |
| Repeat | | This instruction is reported then it will be execute will reflect the final resources. When repeate 1 during each repetition | eatable. If the operation follows a RPT instruction, d N+1 times. The state of the Z, N, C and OVC flags sult. The V flag will be set if an intermediate overflow ed, the program-memory address is incremented by on. |
| Example | ; Calc; ; int1 ; int1 ; sum ; for(; { ; s ; x ; X ; } | <pre>ulate FIR filter us 6 X[N] ; Data in 6 C[N] ; Coeffic = X[N-1] * C[0]; i=1; i < N; i++) um = sum + (X[N-1-i [N-i] = X[N-1-i]; x[0].</pre> | <pre>ing 16-bit multiply: formation ient information, located in high 64K] * C[i]) >> 5;</pre> |
| | ; X[1] | = X[0]; | WARD we have be used of Warman |
| | | XARZ, #X+N | ; $AR2 = point to end of x array$ |
| | JPM ZADA | -5 | · Zero ACC P OVC |
| | XMAC | P *XAR2 *(C) | ACC = 0 P = X[N-1] * C[0] |
| | RPT | #N-2 | : Repeat next instruction N-1 times |
| | XMAC | D P,*XAR2,*(C+1) | <pre>; ACC = ACC + P >> 5, ; P = X[N-1-i] * C[i], ; i++</pre> |
| | MOV | *+XAR2[2],T | ; $X[1] = X[0]$ |
| | ADDL | ACC,P << PM | ; Perform final accumulate |
| | MOVL | @sum,ACC | ; Store final result into sum |

| XOR ACC,lo | c16 | | | Bitwise E | Exclusi | /e OR | |
|--|-------------------------------|---|---|---|-------------|-------|--|
| S | YNTAX | OPTIONS | OPCODE | OBJMODE | RPT | CYC | |
| XOR ACC,loc1 | 6 | | 1011 0111 LLLL LLLL | 1 | Y | N+1 | |
| Operands | ACC loc16 | Accumulator registe Addressing mode (s | Accumulator register Addressing mode (see Chapter 5) | | | | |
| Description | | Perform a bitwise zero-extended conte mode. The result is a ACC = ACC XOR 0: | Perform a bitwise XOR operation on the ACC register with the zero-extended content of the location pointed to by the "loc16" address mode. The result is stored in the ACC register: ACC = ACC XOR 0: [loc16]; | | | | |
| Flags and Modes | N Z | The load to ACC is to the negative flag bit The load to ACC is t operation generates | The load to ACC is tested for a negative condition. If bit 31 of ACC is 1, then the negative flag bit is set; otherwise it is cleared. The load to ACC is tested for a zero condition. The zero flag bit is set if the operation generates $ACC = 0$; otherwise it is cleared | | | | |
| RepeatThis operation is repeatable. If the operation follows a RPT instruction, then the XOR instruction will be executed N+1 times. The state of the Z and N flags will reflect the final result. | | | | | | | |
| Example | ; Calc MOVL XOR MOVL | ulate the 32-bit va ACC,@VarA ACC,@VarB @VarA,ACC | alue: VarA = VarA XOR (; Load ACC with con ; XOR ACC with cont ; Store result in V | 0:VarB tents of Va ents of 0:V arA | ırA VarB | | |

XOR ACC,#16bit << #0..16

Bitwise Exclusive OR

| S | SYNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | |
|-----------------------------------|----------------------------------|--|---|--|-----|-----|--|
| XOR ACC,#16 | bit << #0. | 15 | 0011 1110 0010 SHFT CCCC CCCC CCCC CCCC | 1 | - | 1 | |
| XOR ACC,#16bit << #16 | | | 0101 0110 0100 1110 CCCC CCCC CCCC CCCC | 1 | - | 1 | |
| Operands | ACC | Accumulator registe | r | | | | |
| | #16bit | 16-bit immediate co | nstant value | | | | |
| | #016 | Shift value (default i | s "<< #0" if no value specifi | ed) | | | |
| Description Flags and Modes | N | Perform a bitwise X0 unsigned constant va and lower order bits stored in the ACC re ACC = ACC XOR (0 The load to ACC is to the negative flag bit | Perform a bitwise XOR operation on the ACC register with the given 16-bit unsigned constant value left shifted as specified. The value is zero extended and lower order bits are zero filled before the XOR operation. The result is stored in the ACC register: ACC = ACC XOR (0:16bit << shift value); The load to ACC is tested for a negative condition. If bit 31 of ACC is 1, then | | | | |
| | Z | The load to ACC is tested for a zero condition. The zero flag bit is set if the operation generates $ACC = 0$; otherwise it is cleared | | | | | |
| Repeat | | This instruction is not repeatable. If this instruction follows the RPT in- struction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; Calco MOVL XOR MOVL (| ulate the 32-bit v ACC,@VarA ACC,#0x8000 << 12 @VarA,ACC | alue: VarA = VarA XOR (; Load ACC with con ; XOR ACC with 0x08 ; Store result in V | 0x08000000 tents of Va 0000000 VarA | ırA | | |
XOR AX,loc16

Bitwise Exclusive OR

| s | YNTAX C | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
|--------------------|------------------------------|---|--|--|---------------------|---------------|--|--|--|
| XOR AX, loc1 | 6 | | 0111 000A LLLL LLLL | Х | - | 1 | | | |
| Operands | AX | Accumulator high (A | Accumulator high (AH) or accumulator low (AL) register | | | | | | |
| | loc16 | Addressing mode (s | Addressing mode (see Chapter 5) | | | | | | |
| Description | | Perform a bitwise exclusive OR operation on the specified AX register (AH or AL) and the contents of the location pointed to by the "loc16" addressing mode. The result is stored in the specified AX register: AX = AX XOR [loc16]; | | | | | | | |
| Flags and Modes | Ν | The load to AX is tes negative flag bit is se | The load to AX is tested for a negative condition. If bit 15 of AX is 1, then the negative flag bit is set; otherwise it is cleared. | | | | | | |
| | z | The load to AX is ter operation generates | sted for a zero condition. T AX = 0, otherwise it is clea | he zero flag k red. | oit is se | t if the | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the s only o | e RPT nce. | | | |
| Example | ; XOR t MOV XOR MOV | the contents of Van AL,@VarA AL,@VarB @VarC,AL | rA and VarB and store : ; Load AL with cont ; XOR AL with conte ; Store result in V | in VarC: cents of Var ents of VarB VarC | rA 3 | | | | |

XOR loc16, AX

Bitwise Exclusive OR

| S | Ο ΧΑΤΝΥ | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | | |
|--------------------|-----------------------|--|--|---------------------------------|---------------------|----------------|--|--|--|--|
| XOR loc16, AX | κ | | 1111 001A LLLL LLLL | Х | - | 1 | | | | |
| Operands | loc16 | Addressing mode (s | Addressing mode (see Chapter 5) | | | | | | | |
| | AX | Accumulator high (A | ccumulator high (AH) or accumulator low (AL) register | | | | | | | |
| Description | | Perform a bitwise exclusive OR operation on the 16-bit contents of locati pointed to by the "loc16" addressing mode and the specified AX register (<i>i</i> or AL). The result is stored in the location pointed to by "loc16": | | | | | | | | |
| | | [loc16] = [loc16] | [loc16] = [loc16] XOR AX; | | | | | | | |
| | | This instruction perfe | orms a read-modify-write op | peration. | | | | | | |
| Flags and Modes | N | The load to [loc16] is then the negative fla | s tested for a negative condi Ig bit is set; otherwise it is c | tion. If bit 15 leared. | of [loc1 | 6] is 1, | | | | |
| | Z | The load to [loc16] is operation generates | tested for a zero condition. [loc16] = 0, otherwise it is o | The zero flag cleared. | bit is se | et if the | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this inst the repeat counter (RPTC) | struction follo and executes | ows the s only o | → RPT ince. | | | | |
| Example | ; XOR t MOV XOR | the contents of Va AL,@VarA @VarB,AL | Provide and store in VarB: AL,@VarA ; Load AL with contents of VarA VarB,AL ; VarB = VarB XOR AL | | | | | | | |

XOR loc16,#16bit

Bitwise Exclusive OR

| S | YNTAX OI | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | | |
|---------------|----------------------------|---|--|----------------|-------|-----|--|--|--|--|
| XOR loc16,#16 | Bbit | | 0001 1100 LLLL LLLL CCCC CCCC CCCC CCCC | х | - | 1 | | | | |
| Operands | loc16 | Addressing mode (s | Addressing mode (see Chapter 5) | | | | | | | |
| | #16bit | 16-bit immediate cor | nstant value | | | | | | | |
| Description | | Perform a bitwise XC the "loc16" addressir result is stored in the | erform a bitwise XOR operation on the content of the location pointed to by e "loc16" addressing mode and the 16-bit immediate constant value. The sult is stored in the location pointed to by "loc16": | | | | | | | |
| | | [loc16] = [loc16] | oc16] = [loc16] XOR 16bit; | | | | | | | |
| | | Smart Encoding: If loc16 = AH or AL encode this instructi the XORW AX,#16b | Smart Encoding: f loc16 = AH or AL and #16bit is an 8-bit number, then the assembler will encode this instruction as XO"RB AX,#8bt. To override this encoding, use he XORW AX,#16bit instruction alias. | | | | | | | |
| Flags and | Ν | After the operation if | bit 15 of [loc16] 1, set N; o | therwise, clea | ar N. | | | | | |
| Modes | z | After the operation if | After the operation if [loc16] is zero, set Z; otherwise, clear Z. | | | | | | | |
| Repeat | | This instruction is instruction, it resets | This instruction is not repeatable. If this instruction follows the RPT nstruction, it resets the repeat counter (RPTC) and executes only once. | | | | | | | |
| Example | ; Toggl ; VarA XOR @ | e Bits 2 and 14 o: = VarA XOR #(1 << VarA,#(1 << 2 1 | Bits 2 and 14 of VarA: VarA XOR #(1 << 2 1 << 14) rA,#(1 << 2 1 << 14) ; Toggle bits 2 and 11 of VarA | | | | | | | |

XORB AX, #8bit

Bitwise Exclusive OR 8-bit Value

| S | YNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | |
|--------------------|-------------------------------|---|--|--|--------------------|-------|--|--|
| XORB AX, #8b | oit | | 1111 000A CCCC CCCC | Х | - | 1 | | |
| Operands | AX #8bit | Accumulator high (A 8-bit immediate cons | Accumulator high (AH) or accumulator low (AL) register 3-bit immediate constant value | | | | | |
| Description | | Perform a bitwise ex the 8-bit unsigned im the AX register: AX = AX XOR 0x00: | Perform a bitwise exclusive OR operation on the specified AX register and the 8-bit unsigned immediate constant zero extended. The result is stored in the AX register: AX = AX XOR 0x00:8bit; | | | | | |
| Flags and Modes | N Z | The load to AX is tes negative flag bit is so The load to AX is te operation generates | The load to AX is tested for a negative condition. If bit 15 of AX is 1, then the negative flag bit is set; otherwise it is cleared. The load to AX is tested for a zero condition. The zero flag bit is set if the operation generates [loc16] = 0, otherwise it is cleared. | | | | | |
| Repeat | | This instruction is instruction, it resets | not repeatable. If this ins the repeat counter (RPTC) | struction follo and executes | ows the sonly o | e RPT | | |
| Example | ; Toggl MOV XORB MOV | e bit 7 of VarA a AL,@VarA AL,#0x80 @VarB,AL | nd store result in Var ; Load AL with cont ; XOR contents of A ; Store result in V | 3: cents of Var AL with 0x00 VarB | rA)80 | | | |

| XPREAD loc | 16, *(pma | a) | C2xLP Source-Compatible Program Read | | | | | | |
|--------------------|---|---|---|--|--|---|---|--|--|
| S | | PTIONS | OPC | ODE | OBJMODE | RPT | CYC | | |
| XPREAD loc16 | δ,*(pma) | | 1010 1100 LLLL LLLL | MMMM MMMM LLLL LLLL | 1 | Y | N+2 | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5 |) | | | | | |
| | *(pma) | Immediate program- space range (0x3F0 | memory addr 000 to 0x3FF | ess, can only a FFF) | access high 64 | 4K of pr | ogram | | |
| Description | | Load the 16-bit data mode with the 16-bit "*(pma)" addressing | -memory loca content of th mode: | ation pointed to e program-me | o by the "loc1 mory location | 6" addr pointe | essing d to by | | |
| | | [loc16] = Prog[0] | c16] = Prog[0x3F:pma]; | | | | | | |
| | | The C28x forces the by the "*(pma)" add XPREAD instruction 64K of program add devices, memory bl (unified memory), h access data space v | upper 6 bits of dressing mod a. This limits t dress space locks are ma ence the "*(p variables that | of the program e, to 0x3F wi he program m (0x3F0000 to upped to both oma)" address fall within its a | memory addr hen using this nemory addres 0x3FFFFF). program and sing mode ca address range | ess, sp s form ss to th On the d data n be u | ecified of the le high C28x space sed to | | |
| Flags and Modes | Ν | If (loc16 = @AX) and | d bit 15 of AX | is 1, then N is | set; otherwise | e N is cl | eared. | | |
| | z | If (loc16 = @AX) an cleared. | d the value o | f AX is zero, t | hen Z is set; c | otherwi | se Z is | | |
| Repeat | | This instruction is rep it will be execut program-memory ad address is post-incre | beatable. If the ted N+1 ti ldress is copi emented by 1 | e operation foll mes. When ed to an interr during each r | lows a RPT in repeated, f nal shadow rep repetition. | struction the "* gister a | n, then (pma)" ınd the | | |
| Example | ; Copy ; int16 ; int16 ; for(i ; Arr MOVL RPT XPREA | <pre>the contents of A: Array1[N]; // Ld Array2[N]; // Ld =0; i < N; i++) ay2[i] = Array1[i] XAR2,#Array2 #(N-1) D *XAR2++,*(Array3)</pre> | rrayl to Ar ocated in h ocated in d]; ; XA ; Re 1) ; Ar ; i+ | ray2: igh 64K of] ata space R2 = pointe peat next i: ray2[i] = A + | program space r to Array2 nstruction 1 rray1[i], | ce N time | S | | |

| XPREAD loc16, *AL C2xLP Source-Compatible Program Read | | | | | | Read | | | |
|--|---|---|---|--|------------------------------|-----------------------------|--|--|--|
| Ş | SYNTAX O | PTIONS | OPCODE | OBJMODE | RPT | CYC | | | |
| XPREAD loc1 | 6,*AL | | 0101 0110 0011 1100 0000 0000 LLLL LLLL | 1 | Y | N+4 | | | |
| Operands | loc16 | Addressing mode (s | ee Chapter 5) | | | | | | |
| | *AL | Indirect program-me can only access high | ndirect program-memory addressing using register AL, an only access high 64K of program space range (0x3F0000 to 0x3FFFFF | | | | | | |
| Description | | Load the 16-bit data mode with the 16-bit "*AL" addressing mo | Load the 16-bit data-memory location pointed to by the "loc16" addressing mode with the 16-bit content of the program-memory location pointed to by "*AL" addressing mode: | | | | | | |
| | | [loc16] = Prog[02 | x3F:AL]; | | | | | | |
| | | The C28x forces the by the "*AL" address instruction. This lin program address sp memory blocks are memory), hence the space variables that | The C28x forces the upper 6 bits of the program memory address, specified by the "*AL" addressing mode, to 0x3F when using this form of the XPREAD instruction. This limits the program memory address to the high 64K of program address space (0x3F0000 to 0x3FFFFF). On the C28x devices, memory blocks are mapped to both program and data space (unified memory), hence the "*AL" addressing mode can be used to access data space variables that fall within its address range. | | | | | | |
| Flags and Modes | N | If (loc16 = @AX) and | d bit 15 of AX is 1, then N is | set; otherwise | e N is c | leared. | | | |
| | Z | lf (loc16 = @AX) an cleared. | d the value of AX is zero, t | hen Z is set; o | otherwi | se Z is | | | |
| Repeat | | This instruction is rep it will be executed N address is copied post-incremented by | peatable. If the operation foll I+1 times. When repeated, to an internal shadow re / 1 during each repetition. | lows a RPT in the "*AL" proo gister and th | structio gram-m e addr | n, then iemory ess is | | | |
| Example | ; Copy ; int16 ; int16 ; for(i ; Arr MOV MOVL RPT XPREA | the contents of A Array1[N]; // L Array2[N]; =0; i < N; i++) ay2[i] = Array1[i @AL,#Array1 XAR2,#Array2 #(N-1) D *XAR2++,*AL | <pre>rray1 to Array2: ocated in high 64K of p // Located in data space]; ; AL = pointer to ; XAR2 = pointer to ; Repeat next instru ; Array2[i] = Array1 ; i++</pre> | program space Array1 Array2 action N tin .[i], | nes | | | | |

| XPWRITE *A | ,loc16 | | C2xLP Source-Compatible Program Write | | | | | |
|--------------------|---|---|---|---|---|--|--|--|
| Ş | SYNTAX O | PTIONS | 0 | PCODE | | OBJMODE | RPT | CYC |
| XPWRITE *AL | .,loc16 | | 0101 011 0000 000 | 0 0011 110 0 LLLL LLI |)1 LL | 1 | Y | N+4 |
| Operands | *AL | Indirect program-me high 64K of program | emory add n space rar | ressing usin ige (0x3F00 | ig req 100 to | gister AL, car 0x3FFFFF) | n only a | access |
| | loc16 | Addressing mode (s | see Chapte | r 5) | | | | |
| Description | | Load the 16-bit prog mode with the 16-l addressing mode: | gram-mem bit content | ory location of the loca | poin ation | ted to by "*Al pointed to b | L" addr y the ' | essing 'loc16" |
| | | <pre>Prog[0x3F:AL] =</pre> | [loc16]; | | | | | |
| | | The C28x forces the by the "*AL" address instruction. This lim program address sp memory blocks are memory), hence the space variables that | e upper 6 bit sing mode, t hits the pro pace (0x3F e mapped e "*AL" ado t fall within | s of the prog o 0x3F when gram memo 0000 to 0x3 to both pro lressing mo its address | gram n usin ory a BFFFI ogram de ca range | memory addr g this form of ddress to the FF). On the C a and data s an be used to e. | ress, sp the XP\ e high (228x de pace (paces) | ecified NRITE 64K of evices, unified s data |
| Flags and Modes | | None | | | | | | |
| Repeat | | This instruction is re it will be executed N address is copied post-incremented by | peatable. If I+1 times. ^v to an inte y 1 during e | the operatio When repea rnal shadov each repetiti | on foll ited, t w reg on. | ows a RPT in the "*AL" proo gister and th | structio gram-m e addr | n, then iemory [.] ess is |
| Example | ; Copy ; intle ; intle ; for(i ; Arr MOVL MOV RPT XPWR] | <pre>the contents of A 5 Array1[N]; // L 5 Array2[N]; // L i=0; i < N; i++) ray2[i] = Array1[i</pre> | rrayl to ocated in ocated in ; XAR2 = ; AL = ; Repeat ; Array2 ; i++ | Array2: data spac high 64K pointer t pointer t next inst [i] = Arra | ce of p o Ar o Ar ruct yl[i | program spac ray1 ray2 ion N times], | ce | |

| XRET | ET C2xLP Source-Compatible Retur | | | | | | |
|--------------------|---|--|---|------------------------------|--------------------------------|--|--|
| | SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC | | |
| XRET | | 0101 0110 1111 1111 | 1 | - | 7 | | |
| Note: XRET is | an alias for RETC unconditional. | • | | • | | | |
| Operands | None | | | | | | |
| Description | Return conditional popped from the st upper 6 bits of the with the instruction | ly. If the specified condition tack and stored into the low PC are forced to 0x3F; Othe following the XRETC operation | n is true, a 1 16 bits of the erwise, execut tion: | 6-bit v PC wł tion coi | alue is nile the ntinues | | |
| | if(COND = true) SP = SP - 1; PC = 0x3F:[SI | 2]; | | | | | |
| | Note: This instruction range of progr XCALL, the XF | can transfer program control only to am space (0x3F0000 to 0x3FFFf RET instruction must be used. | a location located -). To return from | d in the u n a call i | pper 64K made by | | |
| Flags and Modes | V If the V flag is teste | ed by the condition, then V is | cleared. | | | | |
| Repeat | This instruction is instruction, it resets | not repeatable. If this in sthe repeat counter (RPTC) | struction follo and executes | ows the | e RPT once. | | |
| Example | ; Return from FuncA if ; ; to zero and return. T ; in upper 64K of progra | VarA does not equal zer his example only works a am space: | o, else set for code lo | VarB cated | | | |
| | XCALL FuncA | ; Call FuncA | | | | | |
| | FuncA: | ; Function A: | | | | | |
| | | | | | | | |
| | MOV AL,@VarA XRET NEQ MOV @VarA,#0 XRETC UNC | ; Load AL with ; Return if Var. ; Store 0 into ; Return uncond | contents of A does not VarB itionally | VarA equal | 0 | | |

XRETC COND

C2xLP Source-Compatible Conditional Return

| SYNTAX OPTIONS | OPCODE | OBJMODE | RPT | CYC |
|----------------|---------------------|---------|-----|-----|
| XRETC COND | 0101 0110 1111 COND | 1 | - | 4/7 |

| Operands | COND | Condit | ional codes: | | |
|--------------------|------|---|--|--|---|
| | | COND | Syntax | Description | Flags Tested |
| | | 0000 | NEQ | Not Equal To | Z = 0 |
| | | 0001 | EQ | Equal To | Z = 1 |
| | | 0010 | GT | Greater Then | Z = 0 AND $N = 0$ |
| | | 0011 | GEQ | Greater Then Or Equal To | N = 0 |
| | | 0100 | LT | Less Then | N = 1 |
| | | 0101 | LEQ | Less Then Or Equal To | Z = 1 OR N = 1 |
| | | 0110 | HI | Higher | C = 1 AND $Z = 0$ |
| | | 0111 | HIS, C | Higher Or Same, Carry Set | C = 1 |
| | | 1000 | LO, NC | Lower, Carry Clear | C = 0 |
| | | 1001 | LOS | Lower Or Same | C = 0 OR Z = 1 |
| | | 1010 | NOV | No Overflow | V = 0 |
| | | 1011 | OV | Overflow | V = 1 |
| | | 1100 | NTC | Test Bit Not Set | TC = 0 |
| | | 1101 | TC | Test Bit Set | TC = 1 |
| | | 1110 | NBIO | BIO Input Equal To Zero | BIO = 0 |
| | | 1111 | UNC | Unconditional | - |
| | | poppe upper with th if (COI { SP PC | d from the st 6 bits of the e instruction ND = true) = SP - 1; = 0x3F:[SE | ack and stored into the low 16 b PC are forced to 0x3F; Otherwise following the XRETC operation: | its of the PC while the e, execution continues |
| | | } else PC | = PC + 1; | | |
| | | Note: | This instruction range of progra XCALL, the XF If (COND = true If (COND = fals | can only transfer program control to a loca am space (0x3F0000 to 0x3FFFF). To IETC instruction must be used. The cycle e) then the instruction takes 7 cycles. se) then the instruction takes 4 cycles. | ation located in the upper 64K return from a call made by times for this operation are: |
| Flags and Modes | V | If the \ | / flag is teste | d by the condition, then V is clea | ired. |
| Repeat | | This in instruction | nstruction is tion, it resets | not repeatable. If this instruc the repeat counter (RPTC) and | tion follows the RPT executes only once. |

| Example | <pre>ample ; Return from FuncA if VarA does not equal zero, else set Va ; to zero and return. This example only works for code locat ; in upper 64K of program space:</pre> | | | | | | | |
|---------|---|----------|---|---------------------------------|--|--|--|--|
| | XCALL | FuncA | ; | Call FuncA | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | FuncA: | | ; | Function A: | | | | |
| | • | | | | | | | |
| | • | | | | | | | |
| | • | | | | | | | |
| | • | | | | | | | |
| | MOV | AL,@VarA | ; | Load AL with contents of VarA | | | | |
| | XRETC | NEQ | ; | Return if VarA does not equal 0 | | | | |
| | MOV | @VarA,#0 | ; | Store 0 into VarB | | | | |
| | XRETC | UNC | ; | Return unconditionally | | | | |

ZALR ACC, loc16

Zero AL and Load AH With Rounding

| S | SYNTAX (| OPTIONS | OPCODE | OBJMODE | RPT | CYC | | | | |
|--------------------|------------------|--|--|---------------------------------|-------------------|---------------|--|--|--|--|
| ZALR ACC,loc | 16 | | 0101 0110 0001 0011 0000 0000 LLLL LLLL | 1 | - | 1 | | | | |
| Operands | ACC | Accumulator register | ccumulator register | | | | | | | |
| Description | 10010 | Load low accumulato (AH) with the 16-bit c | ad low accumulator (AL) with the value 0x8000 and load high accumulator | | | | | | | |
| | | AII = [10010]; AL = 0x8000; | | | | | | | | |
| Flags and Modes | N | The load to ACC is te the negative flag bit is | The load to ACC is tested for a negative condition. If bit 31 of ACC is 1, then he negative flag bit is set; otherwise it is cleared. | | | | | | | |
| | Z | The load to ACC is te operation generates | The load to ACC is tested for a zero condition. The zero flag bit is set if the peration generates $ACC = 0$; otherwise it is cleared | | | | | | | |
| Repeat | | This instruction is instruction, it resets the | not repeatable. If this ins he repeat counter (RPTC) a | struction follo and executes | ws the only or | e RPT ice. | | | | |
| Example | ; Calc ; Y, M | ulate: Y = round(M , X, B are all Q15 | *X << 1 + B << 16) numbers | | | | | | | |
| | SPM | +1 | ; Set product shift mo | de to << 1 | | | | | | |
| | MOV | T,@M | ; $T = M$ | (Q15) | | | | | | |
| | MPY | P,T,@X | ; $P = M * X$, $ACC = R < 16 + 0x^{20}$ | (Q3U) 00 (Q31) | | | | | | |
| | ADDI. | ACC P << PM | ; ALL = $B << 10 + 0000$ • Add P to ACC with ah | ift (031) | | | | | | |
| | MOV | @Y,AH | ; Store AH into Y | (015) | | | | | | |
| | | • | • | · ~ · · / | | | | | | |

| ZAP OVC | | | | Clear Over | flow Co | ounter |
|--------------------|--|---|--|------------------|---------|--------|
| SYNTAX OPTIONS | | | OPCODE | OBJMODE | RPT | CYC |
| ZAP OVC | | | 0101 0110 0101 1100 | 1 | - | 1 |
| Operands | OVC | overflow counter bits in Status Register 0 (ST0) | | | | |
| Description | | Clear the overflow counter (OVC) bits in Status Register 0 (ST0). | | | | |
| Flags and Modes | ovc | The 6-bit overflow counter bits (OVC) are cleared. | | | | |
| Repeat | This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once. | | | | | |
| Example | ; Cal ZAP MOVL ADDL ADDL SAT MOVL | culate: VarD = sat(OVC ACC,@VarA ACC,@VarB ACC,@VarC ACC @VarD,ACC | <pre>VarA + VarB + VarC) ; Zero overflow counte ; ACC = VarA ; ACC = ACC + VarB ; ACC = ACC + VarC ; Saturate if OVC != 0 ; Store saturated resu</pre> | r lt into Var | Đ | |

Zero Accumulator and P Register

| SYNTAX OPTIONS | | | | OPCODE | OBJMODE | RPT | CYC |
|---|---|--|---|--|----------------|--------|-----|
| ZAPA | ZAPA | | | 0110 0011 0011 | 1 | - | 1 |
| Operands | None | | | | | - | |
| Description | ription Zero the ACC and P register ACC = 0; P = 0; OVC = 0; | | | ers as well as the ov | erflow counte | r (OVC | ·): |
| Flags and Modes | and N The N bit is set. s | | | | | | |
| | Z | The Z bit is cleared. | | | | | |
| Repeat | This instruction is not repeatable. If this instruction follows the RF instruction, it resets the repeat counter (RPTC) and executes only once. | | | | ∍ RPT ince. | | |
| Example | <pre>; Calculate sum of product using 32-bit multiply and retain ; high result: ; int32 X[N]; // Data information ; int32 C[N]; // Coefficient information (located in low 4M) ; int32 sum = 0; ; for(i=0; i < N; i++) ; sum = sum + ((X[i] * C[i]) >> 32) >> 5;</pre> | | | | | | |
| MOVL XAR2, #X ; MOVL XAR7, #C ; SPM -5 ; ZAPA ; RPT #(N-1) ; OMACL P.*XAR2++.*XAR7++ ; | | XAR2 = pointer t XAR7 = pointer t Set product shif Zero ACC, P, OVC Repeat next inst ACC = ACC + P >> | to X to C ft to ">> 5' c truction N t > 5, | , cimes | | | |
| | ADDL | ACC, P << PM | ; | <pre>P = (X[i] * C[i] i++ Perform final ad</pre> |) >> 32 | | |
| | MOVL | @sum,ACC | ; | Store final resu | ut into sur | n | |

ZAPA

Chapter 7

Emulation Features

The CPU in the C28x contains hardware extensions for advanced emulation features that can assist you in the development of your application system (software and hardware). This chapter describes the emulation features that are available on all C28x devices using only the JTAG port (with TI extensions).

For more information about instructions shown in examples in this chapter, see Chapter 6, *Assembly Language Instructions*.

| Торі | c Page |
|------|---|
| 7.1 | Overview of Emulation Features7-2 |
| 7.2 | Debug Interface |
| 7.3 | Debug Terminology7-6 |
| 7.4 | Execution Control Modes7-7 |
| 7.5 | Aborting Interrupts With the ABORTI Instruction |
| 7.6 | DT-DMA Mechanism7-16 |
| 7.7 | Analysis Breakpoints, Watchpoints, and Counter(s) |
| 7.8 | Data Logging |

Sharing Analysis Resources7-30

7.10 Diagnostics and Recovery7-31

7.9

7.1 Overview of Emulation Features

The CPU's hardware extensions for advanced emulation features provide simple, inexpensive, and speed-independent access to the CPU for sophisticated debugging and economical system development, without requiring the costly cabling and access to processor pins required by traditional emulator systems. It provides this access without intruding on system resources.

The on-chip development interface provides:

- Minimally intrusive access to internal and external memory
- Minimally intrusive access to CPU and peripheral registers
- Control of the execution of background code while continuing to service time-critical interrupts
 - Break on a software breakpoint instruction (instruction replacement)
 - Break on a specified program or data access without requiring instruction replacement (accomplished using bus comparators)
 - Break on external attention request from debug host or additional hardware
 - Break after the execution of a single instruction (single-stepping)
 - Control over the execution of code from device power up
- Nonintrusive determination of device status
 - Detection of a system reset, emulation/test-logic reset, or powerdown occurrence
 - Detection of the absence of a system clock or memory-ready signal
 - Determination of whether global interrupts are enabled
 - Determination of why debug accesses might be blocked
- Rapid transfer of memory contents between the device and a host (data logging)
- □ A cycle counter for performance benchmarking. With a 100-MHz cycle clock, the counter can benchmark actions up to 3 hours in duration.

7.2 Debug Interface

The target-level TI debug interface uses the five standard IEEE 1149.1 (JTAG) signals (TRST, TCK, TMS, TDI, and TDO) and the two TI extensions (EMU0 and EMU1). Figure 7–1 shows the 14-pin JTAG header that is used to interface the target to a scan controller, and Table 7–1 (page 7-4) defines the pins.

As shown in the figure, the header requires more than the five JTAG signals and the TI extensions. It also requires a test clock return signal (TCK_RET), the target supply (V_{CC}) and ground (GND). TCK_RET is a test clock out of the scan controller and into the target system. The target system uses TCK_RET if it does not supply its own test clock (in which case TCK would simply not be used). In many target systems, TCK_RET is simply connected to TCK and used as the test clock.

Figure 7–1. JTAG Header to Interface a Target to the Scan Controller

| TMS | 1 | 2 | TRST | |
|-----------------------|----|----|--------------|---|
| TDI | 3 | 4 | GND | Header dimensions: Pin-to-pin spacing: 0.100 in. (X.Y) |
| PD (V _{CC}) | 5 | 6 | No pin (key) | Pin width: 0.025-in. square post |
| TDO | 7 | 8 | GND | Pin length: 0.235-in. nominal |
| ICK_RET | 9 | 10 | GND | |
| тск | 11 | 12 | GND | |
| EMU0 | 13 | 14 | EMU1 | |

| Signal | Description | Emulator State [†] | Target State [†] |
|-----------------------|---|--------------------------------|------------------------------|
| EMU0 | Emulation pin 0 | Ι | 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 | Ο |
| ТСК | Test clock. TCK is a clock source from the emulation cable pod. This signal can be used to drive the system test clock. | Ο | I |
| TCK_RET | Test clock return. Test clock input to the emu- lator. Can be a buffered or unbuffered version of TCK. | I | 0 |
| TDI | Test data input | 0 | I |
| TDO | Test data output | I | Ο |
| TMS | Test mode select | 0 | I |
| TRST [‡] | Test reset | 0 | I |

Table 7–1. 14-Pin Header Signal Descriptions

[†] I = input; O = output

[‡] Do not use <u>pullup</u> resistors on TRST: it has an internal pulldown device. In a low-noise environment, 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.)

The state of the TRST, EMU0, and EMU1 signals at device power up determine the operating mode of the device. The operating mode takes effect as soon as the device has sufficient power to operate. Should the TRST signal rise, the EMU0 and EMU1 signals are sampled on its rising edge and the\at operating mode is latched. Some of these modes are reserved for test purposes, but those that can be of use in a target system are detailed in Table 7–2. A target system is not required to support any mode other than normal mode.

| TRST | EMU1 | EMUO | Device Operating Mode | JTAG Cable Active? |
|------|-------------|-------------|--|-----------------------|
| Low | Low | Low | <i>Slave mode.</i> Disables the CPU and memory portions of the C28x. Another processor treats the C28x as a peripheral. | No |
| Low | Low | High | Reserved for testing | No |
| Low | High | Low | <i>Wait-in-reset mode</i> . Prolongs the device's reset until released by ex- ternal means. This allows a C28x to power up in reset, provided external hardware holds EMU0 low only while power-up reset is active. | Yes |
| Low | High | High | Normal mode with emulation dis- abled. This is the setting that should be used on target systems when a scan controller (such as the XDS510) is not attached. TRST will be pulled down and EMU1 and EMU0 pulled up within the C28x; this is the default mode. | No |
| High | Low or High | Low or High | Normal mode with emulation en- abled. This is the setting to use on target systems when a scan control- ler is attached (the scan controller will control TRST). TRST should not be high during device power-up. | Yes |

Table 7–2. Selecting Device Operating Modes By Using TRST, EMU0, and EMU1

7.3 Debug Terminology

The following definitions will help you to understand the information in the rest of this chapter:

- **Background code.** The body of code that can be halted during debugging because it is not time-critical.
- □ **Foreground code.** The code of time-critical interrupt service routines, which are executed even when background code is halted.
- Debug-halt state. The state in which the device does not execute background code.
- ☐ **Time-critical interrupt.** An interrupt that must be serviced even when background code is halted. For example, a time-critical interrupt might service a motor controller or a high-speed timer.
- Debug event. An action, such as the decoding of a software breakpoint instruction, the occurrence of an analysis breakpoint/watchpoint, or a request from a host processor that can result in special debug behavior, such as halting the device or pulsing one of the signals EMU0 or EMU1.
- Break event. A debug event that causes the device to enter the debughalt state.

7.4 Execution Control Modes

The C28x supports two debug execution control modes:

- Stop mode
- Real-time mode

Stop mode provides complete control of program execution, allowing for the disabling of all interrupts. Real-time mode allows time-critical interrupt service routines to be performed while execution of other code is halted. Both execution modes can suspend program execution at break events, such as occurrences of software breakpoint instructions or specified program-space or data-space accesses.

7.4.1 Stop Mode

Stop mode causes break events, such as software breakpoints and analysis watchpoints, to suspend program execution at the next interrupt boundary (which is usually identical to the next instruction boundary). When execution is suspended, all interrupts (including $\overline{\text{NMI}}$ and $\overline{\text{RS}}$) are ignored until the CPU receives a directive to run code again. In stop mode, the CPU can operate in the following execution states:

Debug-halt state. This state is entered through a break event, such as the decoding of a software breakpoint instruction or the occurrence of an analysis breakpoint/watchpoint. This state can also be entered by a request from the host processor. In the stop mode debug-halt state, the CPU is halted. You can place the device into one of the other two states by giving the appropriate command to the debugger.

The CPU cannot service any interrupts, including $\overline{\text{NMI}}$ and $\overline{\text{RS}}$ (reset). When multiple instances of the same interrupt occurs without the first instance being serviced, the later instances are lost.

❑ Single-instruction state. This state is entered when you tell the debugger to execute a single instruction by using a RUN 1 command or a STEP 1 command. The CPU executes the single instruction pointed to by the PC and then returns to the debug-halt state (it executes from one interrupt boundary to the next). The CPU is only in the single-instruction state until that single instruction is done.

If an interrupt occurs in this state, the command used to enter this state determines whether that interrupt can be serviced. If a RUN 1 command was used, the CPU can service the interrupt. If a STEP 1 command was used, the CPU cannot, even if the interrupt is $\overline{\text{NMI}}$ or $\overline{\text{RS}}$.

Run state. This state is entered when you use a run command from the debugger interface. The CPU executes instructions until a debugger command or a debug event returns the CPU to the debug-halt state. The CPU can service all interrupts in this state. When an interrupt occurs simultaneously with a debug event, the debug event has priority; however, if interrupt processing began before the debug event occurred, the debug event cannot be processed until the interrupt service routine begins.

Figure 7–2 illustrates the relationship among the three states. Notice that the C28x cannot pass directly between the single-instruction and run states. Notice also that the CPU can be observed only in the debug-halt state. In practical terms, this means the contents of CPU registers and memory are not updated in the debugger display in the single-instruction state or the run state. Maskable interrupts occurring in any state are latched in the interrupt flag register (IFR).

Figure 7-2. Stop Mode Execution States



[†] If you use a RUN 1 command to execute a single instruction, an interrupt can be serviced in the single-instruction state. If you use a STEP 1 command for the same purpose, an interrupt cannot be serviced.

7.4.2 Real-Time Mode

Real-time mode provides for the debugging of code that interacts with interrupts that must not be disabled. Real-time mode allows you to suspend background code at break events while continuing to execute time-critical interrupt service routines (also referred to as foreground code). In real-time mode, the CPU can operate in the following execution states:

Debug-halt state. This state is entered through a break event such as the decoding of a software breakpoint instruction or the occurrence of an analysis breakpoint/watchpoint. This state can also be enter by a request from the host processor. You can place the device into one of the other two states by giving the appropriate command to the debugger.

In this state, only time-critical interrupts can be serviced. No other code can be executed. Maskable interrupts are considered time-critical if they are enabled in the debug interrupt enable register (DBGIER). If they are also enabled in the interrupt enable register (IER), they are serviced. The interrupt global mask bit (INTM) is ignored. NMI and RS are also considered time-critical, and are always serviced once requested. It is possible for multiple interrupts to occur and be serviced while the device is in the debug-halt state.

Suspending execution adds only one cycle to interrupt latency. When the C28x returns from a time-critical ISR, it reenters the debug-halt state.

If a CPU reset occurs (initiated by $\overline{\text{RS}}$), the device runs the corresponding interrupt service routine until that routine clears the debug enable mask bit (DBGM) in status register ST1. When a reset occurs, DBGM is set, disabling debug events. To reenable debug events, the interrupt service routine must clear DBGM. Only then will the outstanding emulation-suspend condition be recognized.

Note:

Should a time-critical interrupt occur in real-time mode at the precise moment that the debugger receives a RUN command, the time-critical interrupt will be taken and serviced in its entirety before the CPU changes states.

Single-instruction state. This state is entered when you you tell the debugger to execute a single instruction by using a RUN 1 command or a STEP 1 command. The CPU executes the single instruction pointed to by the PC and then returns to the debug-halt state (it executes from one interrupt boundary to the next).

If an interrupt occurs in this state, the command used to enter this state determines whether that interrupt can be serviced. If a RUN 1 command was used, the CPU can service the interrupt. If a STEP 1 command was used, the CPU cannot, even if the interrupt is $\overline{\text{NMI}}$ or $\overline{\text{RS}}$. In real-time mode, if the DBGM bit is 1 (debug events are disabled), a RUN 1 or STEP 1 command forces continuous execution of instructions until DBGM is cleared.

Note: If you single-step an instruction in real-time emulation mode and that instruction sets DBGM, the CPU continues to execute instructions until DBGM is cleared. If you want to single-step through a non-time-critical interrupt service routine (ISR), you must initiate a CLRC DBGM instruction at the beginning of the ISR. Once you clear DBGM, you can single-step or place breakpoints.

Run state. This state is entered when you use a run command from the debugger interface. The CPU executes instructions until a debugger command or a debug event returns the CPU to the debug-halt state.

The CPU can service all interrupts in this state. When an interrupt occurs simultaneously with a debug event, the debug event has priority; however, if interrupt processing began before the debug event occurred, the debug event cannot be processed until the interrupt service routine begins.

Figure 7–3 illustrates the relationship among the three states. Notice that the C28x cannot pass directly between the single-instruction and run states. Notice also that the CPU can be observed in the debug-halt state and in the run state. In the single-instruction state, the contents of CPU registers and memory are not updated in the debugger display. In the debug-halt and run states, register and memory values are updated unless DBGM = 1. Maskable interrupts occurring in any state are latched in the interrupt flag register (IFR).



Figure 7–3. Real-time Mode Execution States

[†] If you use a RUN 1 command to execute a single instruction, an interrupt can be serviced in the single-instruction state. If you use a STEP 1 command for the same purpose, an interrupt cannot be serviced.

Caution about breakpoints within time-critical interrupt service routines

Do not use breakpoints within time-critical interrupt service routines. They will cause the device to enter the debug-halt state, just as if the breakpoint were located in normal code. Once in the debug-halt state, the CPU services requests for $\overline{\text{RS}}$, $\overline{\text{NMI}}$, and those interrupts enabled in the DBGIER and the IER.

After approving a maskable interrupt, the CPU disables the interrupt in the IER. This prevents subsequent occurrences of the interrupt from being serviced until the IER is restored by a return from interrupt (IRET) instruction or until the interrupt is deliberately re-enabled in the interrupt service routine (ISR). Do not reenable that interrupt's IER bit while using breakpoints within the ISR. If you do so and the interrupt is triggered again, the CPU performs a new context save and restarts the interrupt service routine.

7.4.3 Summary of Stop Mode and Real-Time Mode

Figure 7–4 (page 7-12) is a graphical summary of the differences between the execution states of stop mode and real-time mode. Table 7–3 (page 7-13) is a summary of how interrupts are handled in each of the states of stop mode and real-time mode.





[†] If you use a RUN 1 debugger command to execute a single instruction, an interrupt can be serviced in the single-instruction state. If you use a STEP 1 debugger command for the same purpose, an interrupt cannot be serviced.

| Mode | State | If This Interrupt Occurs | The Interrupt Is |
|-----------|--------------------|--------------------------|---|
| Stop | Debug-halt | RS | Not serviced |
| | | NMI | Not serviced |
| | | Maskable interrupt | Latched in IFR but not serviced |
| | Single-instruction | RS | If running: Serviced If stepping: Not serviced |
| | | NMI | If running: Serviced If stepping: Not serviced |
| | | Maskable interrupt | If running: Serviced If stepping: Latched in IFR but not serviced |
| | Run | RS | Serviced |
| | | NMI | Serviced |
| | | Maskable interrupt | Serviced |
| Real-time | Debug-halt | RS | Serviced |
| | | NMI | Serviced |
| | | Maskable interrupt | If time-critical: Serviced. If not time-critical: Latched in IFR but not serviced |
| | Single-instruction | RS | If running: Serviced If stepping: Not serviced |
| | | NMI | If running: Serviced If stepping: Not serviced |
| | | Maskable interrupt | If running: Serviced If stepping: Latched in IFR but not serviced |
| | Run | RS | Serviced |
| | | NMI | Serviced |
| | | Maskable interrupt | Serviced |

Table 7–3. Interrupt Handling Information By Mode and State

Note:

п

L

Unless you are using a real-time operating system, do not enable the realtime operating system interrupt (RTOSINT). RTOSINT is completely disabled when bit 15 in the IER is 0 and bit 15 in the DBGIER is 0.

7.5 Aborting Interrupts With the ABORTI Instruction

Generally, a program uses the IRET instruction to return from an interrupt. The IRET instruction restores all the values that were saved to the stack during the automatic context save. In restoring status register ST1 and the debug status register (DBGSTAT), IRET restores the debug context that was present before the interrupt.

In some target applications, you might have interrupts that must not be returned from by the IRET instruction. Not using IRET can cause a problem for the emulation logic, because the emulation logic assumes the original debug context will be restored. The abort interrupt (ABORTI) instruction is provided as a means to indicate that the debug context will not be restored and the debug logic needs to be reset to its default state. As part of its operation, the ABORTI instruction:

- Sets the DBGM bit in ST1. This disables debug events.
- Modifies select bits in DBGSTAT. The effect is a resetting of the debug context. If the CPU was in the debug-halt state before the interrupt occurred, the CPU does not halt when the interrupt is aborted. Teh CPU automatically switches to the run state. If you want to abort an interrupt, but keep the CPU halted, insert a breakpoint after the ABORTI instruction.

The ABORTI instruction does not modify the DBGIER, the IER, the INTM bit, or any analysis registers (for example, registers used for breakpoints, watchpoints, and data logging).

7.6 DT-DMA Mechanism

The debug-and-test direct memory access (DT-DMA) mechanism provides access to memory, CPU registers, and memory-mapped registers (such as emulation registers and peripheral registers) without direct CPU intervention. DT-DMAs intrude on CPU time; however, you can block them by setting the debug enable mask bit (DBGM) in ST1.

Because the DT-DMA mechanism uses the same memory-access mechanism as the CPU, any read or write access that the CPU can perform in a single operation can be done by a DT-DMA. The DT-DMA mechanism presents an address (and data, in the case of a write) to the CPU, which performs the operation during an unused bus cycle (referred to as a *hole*). Once the CPU has obtained the desired data, it is presented back to the DT-DMA mechanism. The DT-DMA mechanism can operate in the following modes:

- Nonpreemptive mode. The DT-DMA mechanism waits for a hole on the desired memory buses. During the hole, the DT-DMA mechanism uses them to perform its read or write operation. These holes occur naturally while the CPU is waiting for newly fetched instructions, such as during a branch.
- Preemptive mode. In preemptive mode, the DT-DMA mechanism forces the creation of a hole and performs the access.

Nonpreemptive accesses to zero-wait-state memory take no cycles away from the CPU. If wait-stated memory is accessed, the pipeline stalls during each wait state, just as a normal memory access would cause a stall. In realtime mode, DT-DMAs to program memory cannot occur when application code is being run from memory with more than one wait state.

DT-DMAs can be polite or rude.

- **Polite accesses.** Polite DT-DMAs require that DBGM = 0.
- **Rude accesses.** Rude DT-DMAs ignore DBGM.

Figure 7–5 summarizes the process for handling a request from the DT-DMA mechanism.



Figure 7-5. Process for Handling a DT-DMA Request

Some key concepts of the DT-DMA mechanism are:

- Even if DBGM = 0, when the mechanism is in nonpreemptive mode, it must wait for a hole. This minimizes the intrusiveness of the debug access on a system.
- Real-time-mode accesses are typically polite (although there may be reasons, such as error recovery, to perform rude accesses in real-time mode). If the DBGM bit is permanently set to 1 due to a coding bug but you need to regain debug control, use rude accesses, which ignore the state of DBGM.
- In stop mode, DBGM is ignored, and the DT-DMA mode is set to preemptive. This ensures that you can gain visibility to and control of your system if an otherwise unrecoverable error occurs (for example, if ST1 is changed to an undesired value due to stack corruption).

- □ The DT-DMA mechanism does not cause a program-flow discontinuity. No interrupt-like save/restore is performed. When a preemptive DT-DMA forces a hole, no program address counters increment during that cycle.
- □ A DT-DMA request awakens the device from the idle state (initiated by the IDLE instruction). However, unlike returning from an interrupt, the CPU returns to the idle state upon completion of the DT-DMA.

Note:

The information shown on the debugger screen is gathered at different times from the target; therefore, it does not represent a snapshot of the target state, but rather a composite. It also takes the host time to process and display the data. The data does not correspond to the current target state, but rather, the target state as of a few milliseconds ago.

7.7 Analysis Breakpoints, Watchpoints, and Counter(s)

All C28x devices include two analysis units AU1 and AU2. Analysis Unit 1 (AU1) counts events or monitors address buses. Analysis Unit 2 (AU2) monitors address and data buses. You can configure these two analysis units as analysis breakpoints or watchpoints. In addition, AU1 can be configured as a benchmark counter or event counter.

This section describes thee types of analysis features: analysis breakpoints, watchpoints, and counters. Typical analysis unit configurations are presented in section 7.7.4. Data logging is described in section 7.8.

7.7.1 Analysis Breakpoints

An analysis breakpoint is sometimes called a hardware breakpoint, because it acts like a software breakpoint instruction (in this case, the ESTOP0 instruction) but does not require a modification to the application software. An analysis breakpoint triggers a debug event when an instruction at a breakpoint address would have entered the decode 2 phase of the pipeline; this halts the CPU before the instruction is executed. A bus comparator watches the program address bus, comparing its contents against a reference address and a bit mask value.

Consider the following example. If a hardware breakpoint is set at T0, the CPU stops after returning from the T1 subroutine, with the instruction counter (IC) pointing to T0.

| NOP | |
|---------|--------------|
| CALL | T1 |
| ro:movb | AL, #0x00 |
| SB | TIMINGS, UNC |
| L1:NOP | |
| RET | |
| Γ2:NOP | |
| | |

Hardware breakpoints allow masking of address bits. For example, a hardware breakpoint could be placed on the address range 00 0200_{16} -00 $02FF_{16}$ by specifying the following mask address, where the eight LSBs are don't cares:

00 0000 0000 0010 XXXX XXXX₂

7.7.2 Watchpoints

A hardware watchpoint triggers a debug event when either an address or an address and data match a compare value. The address portion is compared against a reference address and bit mask, and the data portion is compared against a reference data value and a bit mask.

When comparing two addresses, you can set two watchpoints. When comparing an address and a data value, you can set only one watchpoint. When performing a read watchpoint, the address is available a few cycles earlier than the data; the watchpoint logic accounts for this.

The point where execution stops depends on whether the watchpoint was a read or write watchpoint, and whether it was an address or an address/data read watchpoint. In the following example, a read address watchpoint occurs when the address X is accessed, and the CPU stops with the instruction counter (IC) pointing three instructions after that point:

```
MOV AR4,#X
MOV AL,*+AR4[0] ; Data read
nop
nop
nop ; The IC will point here
```

For a read watchpoint that requires both an address and data match, the CPU stops with the IC pointing six instructions after that point:

```
MOV AR4,#X
MOV AL,*+AR4[0] ; Data read
nop
nop
nop
nop
nop
nop
nop
; The IC will point here
```

In the following example, a write address watchpoint occurs when the address Y is accessed, and the CPU stops with the IC pointing six instructions after that point:

```
MOV AR4,#Y
MOV *+AR4[0],AL ; Data write
nop
nop
nop
nop
nop
nop
nop
; The IC will point here
```

7.7.3 Benchmark Counter/Event Counter(s)

The 40-bit performance counter on the C28x can be used as a benchmark counter to increment every CPU clock cycle (it can be configured not to count when the CPU is in the debug-halt state). Wait states affect the counter. Wait states in the read 1 and write pipeline phases of an executing instruction affect the counter, regardless of whether an instruction is being single-stepped or run. However, wait states in the fetch 1 pipeline phase do not affect the counter during single-stepping, because the cycle counting does not begin until the de-

code 2 pipeline phase. The counter counts wait states caused by instructions that are fetched but not executed. In most cases, these effects cancel each other out. Benchmarking is best used for larger portions of code. Do not rely heavily on the precision of the benchmarking. (For more information about the pipeline, see Chapter 4.)

Alternatively, you can configure the 40-bit performance counter as two 16-bit or one 32-bit event counter if you want to generate a debug event when the counter equals a match value. The comparison between the counter value and the match value is done before the count value is incremented. For example, suppose you initialize a counter to 0. A match value of 0 causes an immediate debug event (when the action to be counted occurs), and the counter holds 1 afterward.

You can also clear the counter when a hardware breakpoint or address watchpoint occurs. With this feature, you can implement a mechanism similar to a watchdog timer: if a certain address is not seen on the address bus within a certain number of CPU clock cycles, a debug event occurs.

7.7.4 Typical Analysis Unit Configurations

Each analysis unit can be configured to perform one analysis job at a time. Typical configurations for these two analysis units can be any one of the following:

Two analysis breakpoints (i.e., hardware breakpoints)

Detect when an instruction is executed from a specified address or range of addresses. Each hardware breakpoint only requires one analysis unit.

Two hardware address watch points

Detect when any value is either read from or written to a specified address or a range of addresses. In this case, the data written or read is not specified. Only the address of the location is specified and whether to watch for reads or writes to that address. Each watchpoint only requires one analysis unit.

One address with data watchpoint

Detect when a specified data value is either read from or written to a specified address. In this configuration you can either watch for a read or a write but not both reads and writes. This type of watchpoint requires both analysis units.

A set of two chained breakpoints

Detect when a given instruction is executed after another specified instruction. A benchmark counter/event counter

The benchmark counter is only available with analysis unit 1. This counter can be used as a benchmark counter to count cycles or instructions. It can also be used to count AU2 events.

Configuration of the analysis resources is supported in Code Composer Studio. For more information on configuring these, use the Code Composer online help.

7.8 Data Logging

Data logging enables the C28x to send selected memory values to a host processor using the standard JTAG port and an XDS510 or other compatible scan controller. You control data logging activity with your application code.

To perform data logging, you must create a linear buffer of 32-bit words to hold a packet of information. Your application code controls the size, format, and location of this buffer and also determines when to send a buffer's contents to the host. You can control the size of a data logging buffer in two ways:

- Specify a count value in the upper eight bits of ADDRH (when the number of 32-bit words you want to log is between 1 and 256)
- Specify an end address

Note:

When the debugger is not active, the data logging transfers are considered complete as soon as they are enabled to prevent the application software from getting stuck when there is nothing to receive the data.

7.8.1 Creating a Data Logging Transfer Buffer

To create a data logging transfer buffer, follow these steps in your application code:

- 1) Execute the EALLOW instruction to enable access to emulation registers.
- Specify the start address of the buffer in ADDRL and the six LSBs of ADDRH (see Figure 7–6 and Figure 7–7). The address in ADDRL and ADDRH is called the transfer address.
- 3) Use either of the following methods to specify when data logging is to end:
 - a) If the number of words you want to log is between 1 and 256, specify a count value in the upper eight bits of ADDRH (see Figure 7–7). The form of the count value is 256–*n*, where *n* is the number of 32-bit words you want to log. As each word is transferred, both the transfer address and the count value are decremented.
 - b) If the number of words you want to log is greater than 256, specify a data logging end address in REFL and the six LSBs of REFH (see Figure 7–8 and Figure 7–9). Load the ten MSBs of REFH with 0s. When using this method, be sure to set the data logging end address control register (EVT_CNTRL) first, and then the DMA control register
(DMA_CNTRL). EVT_CNTRL is described in Table 7–5 (page 7-26), and DMA_CNTRL is described in Table 7–4 (page 7-25).

Note:

The application must *not* read from the end address of the buffer during the data logging operation. When the end address appears on the address bus, the C28x ends the transfer.

4) Execute the EDIS instruction to disable access to emulation registers.

See Table 7–4 and Table 7–5 on the following pages for descriptions of the registers associated with data logging.

Figure 7–6. ADDRL (at Data-Space Address 00 0838₁₆)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|-------|----------|--------|-------|----|---|---|---|---|---|
| | | | | | 16 | S LSB | s of tra | ansfer | addre | SS | | | | | |

Figure 7–7. ADDRH (at Data-Space Address 00 0839₁₆)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|--------|--------|----|---|---|------|-------|---|------|----------|----------|-------|----|
| | | ١ | Word o | counte | ər | | | Rese | erved | 6 | MSBs | s of tra | Insfer a | addre | SS |

Figure 7–8. REFL (at Data-Space Address 00 084A₁₆)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|-------|-------|-------|--------|---|---|---|---|---|---|
| | | | | | | 16 LS | Bs of | end a | ddress | ; | | | | | |

Figure 7–9. REFH (at Data-Space Address 00 084B₁₆)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|-------|---------|--------|--------|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 6 MSI | Bs of e | end ad | Idress | |

| Address | Name | Access | Description | | | | | | | |
|-----------------------|-----------|--------|--|--|--|--|--|--|--|--|
| 00 0838 ₁₆ | ADDRL | R/W | Start address register (lower 16 bits) 15:0 Lower 16 bits of start address | | | | | | | |
| 00 0839 ₁₆ | ADDRH | R/W | Word counter/start address register (upper 6 bits) 15:8 Word counter. When using this to stop the data logging transfer, set the counter to 256 - n, where n is the number of 32-bit words to transfer. Otherwise set the counter to 0. 7:6 Reserved. Set to 0. 5:0 Upper 6 bits of start address | | | | | | | |
| 00 083E ₁₆ | DMA_CNTRL | R/W | DMA control register 15:14 Set to 0 13 Set to 1 12 Set to 1 11 Give higher priority to: 0: CPU (nonpreemptive mode) 1: Data logging (preemptive mode) 10 Allow data logging during time-critical ISR? 0: No 1: Yes 9 Allow data logging while DBGM = 1? 0: No (polite accesses) 1: Yes (rude accesses) 8:6 Set to 1 5:4 0: EMU0/EMU1 using TCK 1: EMU0/EMU1 using FCK/2 2: JTAG signals 3: Reserved 3:2 Method for ending data logging session: 0: Use the count register to stop data logging 1: Use an end address to stop data logging 1: Use an end address to stop data logging 1: Claim resource for data logging operation 1: Claim resource for data logging operation 2: Enable resource for data logging operation 3: Data logging operation is complete. Bits 14:10 are corrupted when this occurs. | | | | | | | |
| 00 083F ₁₆ | DMA_ID | R | DMA ID register15:14Resource control: 0: Resource is free 1: Application owns resource 2: Debugger owns resource13:12Set to 3. 11:011:0Set to 1. | | | | | | | |

Table 7-4. Start Address and DMA Registers

| Address | Name | Access | Description |
|-----------------------|-----------|--------|--|
| 00 0848 ₁₆ | MASKL | R/W | Set to 0 |
| 00 0849 ₁₆ | MASKH | R/W | Set to 0 |
| 00 084A ₁₆ | REFL | R/W | Data logging end reference address (lower 16 bits) 15:0 Lower 16 bits of start address |
| 00 084B ₁₆ | REFH | R/W | Data logging end reference address (upper 6 bits)15:6Set to 05:0Upper 6 bits of start address |
| 00 084E ₁₆ | EVT_CNTRL | R/W | Data logging end address control register15:14Set to 013Set to 112Set to 111:5Set to 04:2Set to 11:0End-address resource control/status:0: Release end-address resource.1: Claim end-address resource.2: Enable end-address resource.3: Data logging operation has ended. Bits 14:10 are corrupted when this occurs. |
| 00 084F ₁₆ | EVT_ID | R | Data logging end address ID register15:14Resource control: 0: Resource is free 1: Application owns resource 2: Debugger owns resource13:12Set to 1 11:011:0Set to 2 |

Table 7–5. End-Address Registers

7.8.2 Accessing the Emulation Registers Properly

Make sure your application code follows the following protocol when accessing the emulation registers that have been provided for data logging. Each resource has a control register and an ID register.

- 1) Enable writes to memory-mapped registers by using the EALLOW instruction.
- Write to the appropriate control register to claim the resource you want to use. The resource for data logging transfers uses DMA_CNTRL (see Table 7–4 on page 7-25). The resource for detecting the data logging end address uses EVT_CNTRL (see Table 7–5).

- 3) Wait at least three cycles so that the write to the control register (done in the write phase of the pipeline) occurs before the read from the ID register in step 4. You can fill in the extra cycles with NOP (no operation) instructions or with other instructions that do not involve accessing the emulation registers.
- 4) Read the appropriate ID register and verify that the application is the owner. The resource for data logging transfers uses DMA_ID (see Table 7–4 on page 7-25). The resource for detecting the data logging end address uses EVT_ID (see Table 7–5 on page 7-26). If the application is not the owner, then go back to step 2 until this succeeds (you may want a time-out function to prevent an endless loop). This step is optional. The application would fail to become the owner only if the debugger already owns the resource.
- 5) If the application is the owner, the remaining registers for that function can be programmed, and the control register written to again, to enable the function. However, if the application is not the owner, then all of its writes are ignored.
- 6) Disable writes to memory-mapped emulation registers by executing the EDIS instruction.

If an interrupt occurs between the EALLOW instruction in step 1 and the EDIS instruction in step 6, access to emulation registers are automatically disabled by the CPU before the interrupt service routine begins and automatically reenabled when the CPU returns from the interrupt. This means that there is no need to disable interrupts between the EALLOW instruction and the EDIS instruction.

The debugger can, at your request, seize ownership of a register from the application; however, that is not the normal mode of operation.

7.8.3 Data Log Interrupt (DLOGINT)

The completion of a data logging transfer (determined either by the word counter or by the end address) triggers a DLOGINT request. DLOGINT is serviced only if it is properly enabled. If the CPU is halted in real-time mode, DLO-GINT must be enabled in both the DBGIER and the IER. Otherwise, DLOGINT must be enabled in the IER and by the INTM bit in status register ST1.

This interrupt capability is most useful when there are multiple buffers of data to be transferred through data logging and the completion of one transfer should begin the next.

7.8.4 Examples of Data Logging

Example 7–1 shows how to log 20 32-bit words, starting at address 00 0100₁₆ in data memory. The accesses are preemptive (they have higher priority than the CPU) and rude (they ignore the state of the DBGM bit). In addition, data logging can occur during time-critical interrupt service routines. The application can determine whether the data logging operation is complete by polling the LSB of the DMA control register (DMA_CNTRL) at 00 083E₁₆. When the operation is complete, that bit is set to 1.

Example 7–1. Initialization Code for Data Logging With Word Counter

```
; Base addresses
ADMA
        .set
                0838h
; Offsets
DMA ADDRL .set 0
DMA ADDRH .set 1
DMA_CNTRL .set
                6
DMA_ID
        .set
                7
EALLOW
MOV
      AR4, #ADMA
                                ; AR4 pointing to register base addr
      *+AR4[#DMA_CNTRL],#1
MOV
                                ; Attempt to claim resource
NOP
NOP
NOP
      *+AR4[#DMA ID],#7001h
                               ; Value expected in ID register
CMP
                                ; If we don't see the correct ID, then we
В
      FAIL, NEQ
                                 ; failed (the resource is already in use)
MOV
      *+AR4[#DMA ADDRL],#0100h ; Set starting address of buffer,
                                 ; and then the count
MOV
      *+AR4 [DMA_ADDRH],#((256 - 20) << 8)
MOV
      *+AR4[DMA CNTRL],#3E62h
EDIS
```

Example 7–2 shows how to log from address 00 0100_{16} to address 00 $02FF_{16}$ in data memory. The accesses are nonpreemptive (they have lower priority than the CPU), and are polite (they are not performed when the DBGM bit is 0). The data logging cannot occur when a time-critical interrupt is being serviced. An end address of 00 $02FF_{16}$ is used to end the transfer. The application must not read from 00 $02FF_{16}$ during the data logging; a read from that address stops the data logging. As in Example 7–1, the application can poll the LSB of DMA_CNTRL for a 1 to determine whether the data logging operation is complete.

Example 7–2. Initialization Code for Data Logging With End Address

```
; Base addresses
ADMA .set 0838h
DEVT
             .set 0848h
; Offsets
DMA ADDRL .set
                     0
DMA ADDRH .set
                      1
DMA CNTRL .set 6
DMA_ID .set 7
MASKL
             .set 0
MASKH
             .set 1
REFL.set2REFH.set3EVT_CNTRL.set6
EVT ID .set 7
EALLOW
        AR5, #DEVT; AR5 pointing to End Address registersAR4, #ADMA; AR4 pointing to Start/Control base*+AR5[#EVT_CNTRL],#1; Attempt to claim End Address*+AR4[#DMA_CNTRL],#1; Attempt to claim Start/Control
MOV
       AR5, #DEVT
MOV
      AR4, #ADMA
MOV
MOV
NOP
NOP
NOP
         *+AR5[#EVT_ID],#5002h ; Value expected in ID register
CMP
         FAIL, NEQ
                                             ; If we don't see the correct ID, FAIL
В
         *+AR4[#DMA_ID],#7001h ; Value expected in ID register
CMP
                                            ; If we don't see the correct ID, FAIL
В
         FAIL, NEQ
        *+AR5[#MASKL],#0 ; Attempt to claim End Address
*+AR5[#MASKH],#0 ; Attempt to claim End Address
*+AR5[#REFL],#02FFh ; Stop data logging at address 0x02FF
*+AR5[#REFH],#0 ; Attempt to claim End Addr
*+AR5[#EVT_CNTRL],# (2 | (1<<2) | (1<<12) | (1<<13) )</pre>
MOV
MOV
MOV
MOV
MOV
MOV
         *+AR4 [#DMA ADDRL], #0100h ; Set buffer start address and then the count
         *+AR4 [DMA ADDRH],#0
MOV
MOV
         *+AR4 [DMA CNTRL],#3066h
EDIS
```

7.9 Sharing Analysis Resources

You can use analysis breakpoints, watchpoints, and a benchmark/event counter through the debugger, and you can use data logging through application code. Table 7–6 lists the analysis resources, and Figure 7–10 shows which resources are available to be used at the same time.

When the application owns analysis resources, they will be cleared (made unowned and set to the completed state) by a reset. When the debugger owns the resources, they are not cleared by reset but by the JTAG test-logic reset. This ensures that when you are using the debugger, the resources can be used even while the target system undergoes a reset.

Table 7-6. Analysis Resources

| Resource | Purpose |
|-----------|---|
| BA0 | Break on contents of program address or memory address bus |
| BA1 | Break on contents of program address or memory address bus |
| BD | Break on contents of program data, memory read data, or memory write data in addition to an address bus |
| Data log | Perform data logging using counter |
| Benchmark | Count CPU cycles |

Figure 7–10. Valid Combinations of Analysis Resources

| | BA0 | BA1 | BD | Data log | Benchmark |
|-----------|------|-----|-----|----------|-----------|
| BA0 | Yes | Yes | No | Yes† | Yes |
| BA1 | Yes | Yes | No | No | No |
| BD | No | No | Yes | No | No |
| Data log | Yes† | No | No | Yes | No |
| Benchmark | Yes | No | No | No | Yes |

[†] The data logging mode that uses the word counter allows this combination, but not the data logging mode that uses the end address (see section 7.8, *Data Logging*).

7.10 Diagnostics and Recovery

Debug registers within the CPU keep track of the state of several key signals. This allows diagnosis of such problems as a floating READY signal, $\overline{\text{NMI}}$ signal, or $\overline{\text{RS}}$ (reset) signal. Should the debug software attempt an operation that does not complete after a certain time-out period (as determined by the debug software), it attempts to determine the probable cause and display the situation to you. You can then abort, correct the situation or allow it to correct itself, or chose to override it.

Such situations include:

- RS being asserted
- A ready signal not being asserted for a memory access
- NMI being asserted
- The absence of a functional clock
- □ The occurrence of a JTAG test-logic-reset

This page intentionally left blank

This page intentionally left blank

Appendix A

Register Quick Reference

For the status and control registers of the '28x, this appendix summarizes:

- Their reset values
- □ The instructions available for accessing them
- The functions of their bits

| Тор | ic Page |
|-----|--|
| A.1 | Reset Values of and Instructions for Accessing the Registers A-2 |
| A.2 | Register FiguresA-3 |

A.1 Reset Values of and Instructions for Accessing the Registers

Table A–1 lists the CPU status and control registers, their reset values, and the instructions that are available for accessing the registers.

| Register | Description | Reset Value | Instructions |
|----------|---------------------------------|----------------------------------|-----------------------|
| ST0 | Status register 0 | 0000 0000 0000 0000 ₂ | PUSH, POP, SETC, CLRC |
| ST1 | Status register 1 | 0000 M000 0000 V011 ₂ | PUSH, POP, SETC, CLRC |
| IFR | Interrupt flag register | 0000 0000 0000 0000 ₂ | PUSH, AND, OR |
| IER | Interrupt enable register | 0000 0000 0000 0000 ₂ | MOV, AND, OR |
| DBGIER | Debug interrupt enable register | 0000 0000 0000 0000 ₂ | PUSH, POP |

Table A–1. Reset Values of the Status and Control Registers

Note: V: Bit 3 of ST1 (the VMAP bit) depends on the level of the VMAP input signal at reset. If the VMAP signal is low, the VMAP bit is 0 after reset; if the VMAP signal is high, the VMAP bit is 1 after reset. For C28x devices that do not pin out VMAP, the signal is tied high internal to the device.

M: Bit 11 of ST1 (the M0M1MAP bit) depends on the level of the M0M1MAP input signal at reset. If the M0M1MAP signal is low, the bit is 0, high bit is 1. For C28x devices that do not pinout MOM1MAP, the signal is tied high internal to the device.

A.2 Register Figures

The following figures summarize the content of the '28x status and control registers. Each figure in this section provides information in this way:

- The value shown in the register is the value after reset.
- Each unreserved bit field or set of bits has a callout that very briefly describes its effect on the processor.
- Each nonreserved bit field or set of bits is labeled with one of the following symbols:
 - R indicates that your software can read the bit field but cannot write to it.
 - R/W indicates that your software can read the bit field and write to it.
- Where needed, footnotes provide additional information for a particular figure.

Figure A–1. Status register ST0

| 15 | 14 | 13 | 12 | 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 |
| | | OVC/ | OVCL | J | | | PM | | V | Ν | Z | С | тс | OVM | SXM |
| F - 1 | | | R/W | | | | ¦ R/ | w | RA | N R∕I | W R/V | V R/V | V R/ | W R/ | W R/W |
| | Ove 0 F 1 O | Negat 0 Neg 1 Neg rflow 1 lag is r | ive fla gative gative f lag reset w dete | ig condit condit | ion fal ion tru | se e | | | | Sig 0 \$ 1 \$ ACC or 0 Resu 1 Over | n-exter Bign ext Bign ext Verflow verflow | ension m ension s ension i mode flow no ode sele | ode suppres mode so | ssed elected | |
| Pro 0 0 0 1 0 1 1 0 1 0 1 1 1 1 | oduc) 0 L) 1 M 0 F 1 F) 0 F 1 F 1 F | t shift Left shi No shif Right s Right s Right s Right s Right s | mode ift by 1 t hift by hift by hift by hift by hift by | 1, sig 2, sig 3, sig 4, sig 5, sig 6, sig | n exte n exte n exte n exte n exte n exte | nded nded nded nded nded nded | | | Tes Ho by Carry 0 Cari 1 Cari | st/contr lds resu TBIT or bit ry not de ry detec | ol flag ilt of tes NORM etected/ etected/ | t perfori instruct borrow row not | med tion detecte detecte | d d | |
| Overf Behav opera Signe Incre Decr Unsig Incre gene Decr gene | flow of ves di tions: d ope ement remer ned of ement erate remer erate | rations s by 1 hts by 1 perations s by 1 a Carr a Carr a Borr a Borr | er ly for s for ea 1 for ea 1 for ea ons (O for AE y 1 for S ow | signed C) ach po ach ne VCU) DD op UB op | l and u sitive o egative eration peratio | overflo overflo over ns that ns that | ed ow; flow. t | Z 0 1 | ero flag Zero co Zero co | ndition 1 | false true | | | | |

Note: For more details about ST0, see section 2.3 on page 2-16.



Figure A-2. Status register ST1, Bits15-8

gram M0 is 0-3FF data and program SP starts at 0x400.



Figure A–3. Status Register ST1, Bits 7–0

[†] These reserved bits are always 0s and are not affected by writes.

[‡] The VMAP bit depends on the level of the VMAP input signal at reset. If the VMAP signal is low, the VMAP bit is 0 after reset; if the VMAP signal is high, the VMAP bit is 1 after reset. For C28x devices that do not pin out the VMAP signal, the signal is tied high internal to the device.

Note: For more details about ST1, see section 2.4 on page 2-34.



Note: For more details about the IFR, see section 3.3.1 on page 3-7.



Figure A-5. Interrupt enable register (IER)

Note: For more details about the IER, see section 3.3.2 on page 3-8.



Figure A-6. Debug interrupt enable register (DBGIER)

Note: For more details about the DBGIER, see section 3.3.2 on page 3-8.

Appendix B

Submitting ROM Codes to TI

This appendix defines the scope of code-customized DSPs and describes the procedures for developing prototype and production units. Information on submitting object code and on ordering customer ROM-coded devices is also included.

| Ιορί | c Page |
|-------------|-----------------------------|
| B .1 | IntroductionB-2 |
| B.2 | Code SubmissionB-4 |
| B.3 | ROM LayoutB-5 |
| B.4 | ROM Code Generation FlowB-6 |

B.1 Introduction

ROM devices offer an attractive low cost alternative to flash devices. In a highvolume application, flash devices may be used to develop, test, refine, and finalize the application code. When the code has been finalized, the code can be submitted to Texas Instruments for masking into the on-chip program ROM. Figure B–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 B-1. TMS320 ROM Code Prototype and Production Flowchart

B.2 Code Submission

ROM codes for 28x devices (in COFF format) may be submitted as an attachment to email or in a 3 ½ inch floppy. Each ROM code submitted is assigned a unique "D-number" in the format DExxxnnn. 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, non-production qualified) manufacturing line, the reliability of which is not fully characterized. Therefore, the anticipated inherent reliability of these prototype units cannot be expressly defined.

Customers must also sign 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.

B.3 ROM Layout

1K OTP-ROM will be reserved for TI internal testing. This space will follow the 1K OTP-ROM meant for the customer. Locations 0x3F7FF8 - 0x3F7FFF will contain the CSM passwords similar to the flash parts.

B.4 ROM Code Generation Flow

Step 1: Submission of code to TI

There are three different possibilities while submitting a code for ROM:

- 1) A single COFF file that contains code for both "customer-OTP" as well as "customer ROM" may be submitted.
- 2) Code could be provided in two different COFF files, one for "customer-OTP" and the other for "customer ROM".
- 3) Code could be provided for "customer ROM" alone and not the "customer-OTP".
- Step 2: Creating the ROM memory image

This is done by first creating a memory array corresponding to the size of the ROM (including the OTP areas) and filling it with 0xFFFF. The memory is then loaded with the customer COFF file(s), TI test code and the D-number.

The numerical portion of the D-number is converted to its hexadecimal equivalent and stored in 0x3F7FF2 & 0x3F7FF3 and 0x3D7BFC & 0x3D7BFD. The symbol "DE" is not converted. For example, for DE121001, the decimal number 121001 will be converted to its hexadecimal equivalent (1D8A9) and stored in the D-number locations. D8A9 will be stored in address "n" and 0001 will be stored in address "n+1". The D-number is stored in both "customer-OTP" as well as "customer ROM" areas.

Step 3: Computation of checksum

The checksum is now computed for the ROM contents (Customer-OTP, TI-OTP and main ROM arrays separately) using the following algorithm:

The contents of an address is XORed with the address and the result is stored in a variable. This is done for the chosen ROM array and the results are added together. The final sum total becomes the checksum for that ROM array. Only the least significant 16-bits of the address are used for checksum computation. Any overflow at the end is ignored and only 32-bits from the end-result are stored. Three unique checksums are computed for the three ROM arrays (Customer-OTP, TI-OTP and main ROM arrays) and stored as shown in the table below.

The following memory zones are used for the purpose of checksum computation:

- Customer-OTP" area, including the D-number. Addresses 0x3D7BFE and 0x3D7BFF, which are eventually used to store the checksum are <u>not</u> used in the computation.
- TI-OTP area containing TI test code. Addresses 0x3D7FFE and 0x3D7FFF, which are eventually used to store the checksum are not used in the computation.
- Customer ROM" area, including the ROM entry-point and CSM passwords. Addresses 0x3F7FF4 and 0x3F7FF5, which are eventually used to store the checksum are <u>not</u> used in the computation.

The computed checksum is written into the corresponding locations (see Table B–1). The image of the ROM is now ready in the PC memory.

| Address | Content |
|-----------------------|---|
| 0x3D7800 | 1K OTP for customer code. (referred to in this document as Customer-OTP) |
| 0x3D7BFB | |
| 0x3D7BFC [†] | Low-word of D-number |
| 0x3D7BFD [†] | High-word of D-number |
| 0x3D7BFE [†] | Low-word of checksum (for Customer-OTP) |
| 0x3D7BFF [†] | High-word of checksum (for Customer-OTP) |
| 0x3D7C00 | 1K OTP for TI test code. (referred to in this document as TI-OTP) |
| 0x3D7FFD | |
| 0x3D7FFE | Low-word of checksum (for TI-OTP) |
| 0x3D7FFF | High-word of checksum (for TI-OTP) |
| 0x3D8000‡ | Start address for customer code in ROM (referred to in this document as Customer-ROM) |
| 0x3F7FF1 | End address for customer code in ROM |
| 0x3F7FF2 [†] | Low-word of D-number |
| 0x3F7FF3 [†] | High-word of D-number |
| 0x3F7FF4 [†] | Low-word of checksum (for Customer-ROM) |

Table B-1. Checksum Computation Memory Locations

| 0x3F7FF5 [†] | High-word of checksum (for Customer-ROM) |
|-----------------------|--|
| 0x3F7FF6 | ROM entry-point (Branch instruction) |
| 0x3F7FF7 | ROM entry-point (Branch instruction) |
| 0x3F7FF8 | CSM passwords |
| | |
| 0x3F7FFF | |
| These addresses are | reserved for the ROM code generation flow and cannot be used by customer code. Using these locations |

[†] These addresses are reserved for the ROM code generation flow and cannot be used by customer code. Using these locations to store the D-number and checksum does not compromise code security.

[‡] The start address for customer code in ROM depends on the part number. While the start address is 0x3D8000 for C2812/C2811, it is 0x3E8000 for C2810. The customer code should provide a branch instruction and the corresponding address at locations 0x3F7FF6 & 0x3F7FF7.

Step 4: Saving the COFF files for gate placement

The ROM image is saved into two COFF files, one for the OTP and the other for the main ROM array. The files are named as follows:

DExxy000_OTP.out - ROM image for the OTP

DExxy000_main.out – ROM image for the main ROM array.

These two files will be sent to the customer for approval.

C2xLP and C28x Architectural Differences

This appendix highlights some of the architecture differences between the C2xLP and the C28x. Not all of the changes are listed here. An emphasis is placed on those changes of which you need to be aware while migrating from a C2xLP-based design to a C28x design. In particular changes in CPU registers and memory map are addressed.

| Topi | c Page |
|------|--|
| C.1 | Summary of Architecture Differences Between C2xLP and C28x C-2 |
| C.2 | Registers C-3 |
| C.3 | Memory Map |

C.1 Summary of Architecture Differences Between C2xLP and C28x

The C28x CPU features many improvements over the C2xLP CPU. A summary of the enhancements is given here.

| Feature | C2xLP | C28x |
|--------------------------|---------------------------------|---------------------------------|
| Program memory space | 64K (16 address signals) | 4M (22 address signals) |
| Data memory space | 64K (16 address signals) | 4G (32 address signals) |
| Number of internal buses | 3 (prog, data-read, data-write) | 3 (prog, data-read, data-write) |
| Addressable word size | 16 | 16/32 |
| Multiplier | 16 bits | 16/32 bits |
| Maskable CPU interrupts | 6 | 14 |

Table C-1. General Features

C.1.1 Enhancements of the C28x over the C2xLP:

- Much higher MHz operation
- □ 32 x 32 MAC
- □ 16 x16 Dual MAC
- 32-bit register file
- 32-bit single-cycle operations
- □ 4M linear program-address reach
- □ 4G linear data-address reach
- Dedicated software stack pointer
- Monitorless real-time emulation
- □ 40–50% better C code efficiency than C2xLP
- 20–30% better assembly code efficiency than C2xLP
- Atomic operation eliminates need to disable/re-enable interrupts
- Extended debugging features (Analysis block, data logging, etc.)
- □ Faster interrupt context save/restore
- More efficient addressing modes
- Unified memory map
- Byte packing and unpacking operations

When you first recompile your C2xLP code set for C28x, you will not be able to take advantage of every enhancement since you are limited by the original source code. Once you begin migrating your code, however, you will quickly begin to take advantage of the full capabilities the C28x offers. See Appendix D for help with migration to C28x.

C.2 Registers

The register modifications to the C2xLP are shown in Figure C–1. Registers that are shaded show the changes or enhancements on the C28x. The italicized names on the left are the original C2xLP names for the registers. The names on the right are the C28x names for the registers.



Figure C-1. Register Changes From C2xLP to C28x

[†]On the C2xLP, IMR and IFR were memory mapped. On the C28x, they are registers.

C2xLP and C28x Architectural Differences C-3

C.2.1 CPU Register Changes

A brief description of the register modifications is given below. For a complete description of each register, see descriptions in the C2xLP and C28x Reference Guides.

| хт | Multiplicand register. The 32-bit multiplicand register is called XT on the C28x. The C2xLP TREG is represented by the upper 16 bits (T). The lower 16 bit area is known as TL. The assembler will also accept TH in place of T for the upper 16 bits of the XT register. |
|----------------|---|
| Ρ | Product register. This register is the same as the C2xLP PREG. You can separately access the high half (PH) or the low half (PL) on the C28x |
| ACC | Accumulator. The size of ACC is the same on the C28x. Access to the register has been enhanced. On C28x, you can access it as two 16-bit registers (AL and AH). |
| SP | Stack Pointer. The SP is new on the C28x. It points directly to the C28x software stack |
| XAR0 – XAR7 | Auxiliary registers. All of the auxiliary registers (XARn) are increased to 32 bits on the C28x. This enables a full 32-bit address reach in data space. Some instructions separately access the low half of the registers (ARn). |
| PC | Program counter. The PC is 22 bits on C28x. On the C2xLP, the PC is 16 bits |
| RPC | Return program counter. The RPC register is new on the C28x. When a call operation is performed, the return address is saved in the RPC register and the old value in the RPC is saved on the stack. When a return operation is performed, the return address is read from the RPC register and the value on the stack is written into the RPC register. The net result is that return operations are faster (4 instead of 8 cycles). This register is only used when certain call and return instructions are used. Normal call and return instructions by- pass this register. |
| IER | Interrupt enable register. The IER is analogous to the Interrupt Mask Register (IMR) on the C2xLP. It performs the same function, however, the name has changed to more appropriately describe the function of the register. Each bit in the register enables one of the maskable interrupts. On the C2xLP, there are six maskable CPU in- terrupts. On the C28x CPU, there are 16 CPU interrupts. On the C2xLP, the IMR was memory mapped. |
| DBGIER | Debug interrupt-enable register. The DBGIER is new on the C28x. It enables interrupts during debug events and allows the processor and debugger to perform real-time emulation. |
| IFR | Interrupt flag register. The IFR functions the same as on the C2xLP. There are more valid bits in this register to accommodate the additional interrupts on the C28x. On the C2xLP, the IFR was memory mapped. |

| ST0/ST1 | Status Registers. The C28x status register bit positions are different compared to the C2xLP. Figure C–3 shows the differences. |
|---------|---|
| DP | Data Page Pointer. On the C2xLP the DP is part of status register |

from 9 to 16 bits.

ST0. The DP on the C28x is a separate register and is increased

C.2.2 Data Page (DP) Pointer Changes

C.2.2.1 C2xLP DP

The direct addressing mode on the C2xLP can access any data memory location in the 64K address range of the device using a 9-bit data page pointer and a 7-bit offset, supplied by the instruction, which is concatenated with the data page pointer value to form the 16-bit data address location. An example C2xLP operation is as follows:

LDP #VarA ; Load DP with page location for VarA LACL VarA ; Load ACC low with contents of VarA

The first instruction initializes the DP register value with the "page" location for the specified variable. Each page is 128 words in size. The assembler/linker automatically resolve the page value by dividing the absolute address of the specified location by 128. For example:

```
If "VarA" address = 0x3456, then the DP value is:
    DP(8:0) = 0x3456/128 = 0x69
```

The next instruction will then calculate the 7-bit offset of the specified variable within the 128-word page. This offset value is then embedded in the address field for that instruction. The assembler/linker automatically resolves the offset value by taking the first 7 bits of the absolute address of the specified location. For example:

```
If "VarA" address = 0x3456, then the 7bit offset value is:
7-bit offset = 0x3456 & 0x007F = 0x56
```

C.2.2.2 C28x DP

The C28x also supports the direct addressing mode using the DP register; however, the following changes and enhancements have been made:

- Supports 22-bit address reach
- DP increased from 9 to 16 bits
- DP is a separate 16-bit register
- □ When AMODE == 0, page size is 64 words and DP(15:0) is used
- When AMODE == 1, page size is 128 words and DP(15:1) is used, bit 0 of DP is ignored

When AMODE == 1, the DP and the direct addressing mode behaves identically to the C2xLP but are enhanced to 22-bit address reach from 16. When

AMODE == 0, the page size is reduced by half. This was done to accommodate other useful addressing modes.

The mapping of the direct addressing modes between the C2xLP and the C28x is as shown in Figure C–2.





Using the previous example, the assembler/linker will initialize the DP and offset values as follows on the C28x:

<u>C2xLP Original Source Mode ("-v28 - m20" mode, AMODE == 1)</u>

LDP #VarA ; DP(15:0) = 0x3456/128 << 1 = 0x00D1 LACL VarA ; 7-bit offset = 0x3456 & 0x007F = 0x56

Equivalent C28x Mnemonics (after C2xLP source is reassembled with the C28x assembler)

MOVZ DP,#VarA ; DP(15:0) = 0x3456/128 << 1 = 0x00D1
MOVU ACC,@@VarA ; 7-bit offset = 0x3456 & 0x007F = 0x56</pre>

<u>C28x Addressing Mode</u> ("-v28" mode, AMODE == 0)

| MOVZ | DP,#VarA | ; | DP(15: | 0) | = | 0x3456/ | 64 | Ł | = | 0x00D1 |
|------|-----------|---|--------|--------|---|---------|----|--------|---|--------|
| MOVU | ACC,@VarA | ; | 6-bit | offset | = | 0x3456 | & | 0x003F | = | 0x16 |

Note: When using C28x syntax, the 128 word data page is indicated by using the double "@@" symbol. The 64 word data page is indicated by the single "@" symbol. This helps the user and assembler to track which mode is being used.

C.2.3 Status Register Changes

Figure C-3. Status Register Comparison Between C2xLP and C28x

C2xLP Status Register ST0

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|-------|----|-------|-------|----|-------|---|---|---|---|-------|---|---|---|---|
| | ARP | | OV | OVM | 1 | INTM | | | | | DP | | | | |
| | R/W-X | | R/W-0 | R/W-X | | R/W-1 | | | | | R/W-X | | | | |

Note: R = Read access; W = Write access; value following dash (-) is value after reset.

C28x Status Register ST0

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|-------|--------|----|----|---|--------|---|-------|-------|-------|-------|-------|-------|-------|
| | | OVC/ | OVCU | | | | PM | | V | Ν | Z | С | TC | OVM | SXM |
| | | R/W-0 | 000000 | | | F | R/W-00 | 0 | R/W-0 |

Note: R = Read access; W = Write access; value following dash (-) is value after reset.

C2xLP Status Register ST1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|-------|----|-------|-------|-------|-------|-------|---|---|---|----|---|---|------|-----|
| | ARB | | CNF | тс | SXM | С | 1 | 1 | 1 | 1 | XF | 1 | 1 | PN | 1 |
| | R/W-X | | R/W-0 | R/W-X | R/W-1 | R/W-1 | R/W-1 | | | | | | | R/W- | -00 |

Note: R = Read access; W = Write access; value following dash (-) is value after reset.

C28x Status Register ST1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----------|--------|------|-------|-------|-------|-------|-------|
| IDLESTAT | EALLOW | LOOP | SPA | VMAP | PAGE0 | DBGM | INTM |
| R-0 | R/W-0 | R–0 | R/W-0 | R/W-1 | R/W-0 | R/W-1 | R/W-1 |

| 15–13 | 12 | 11 | 10 | 9 | 8 |
|---------|-------|---------|----------|---------|-------|
| ARP | XF | M0M1MAP | Reserved | OBJMODE | AMODE |
| R/W-000 | R/W-0 | R-1 | R/W-0 | R/W-0 | R/W-0 |

Notes: 1) R = Read access; W = Write access; value following dash (-) is value after reset; reserved bits are always 0s and are not affected by writes.

| Z | Zero flag. Z is new on the C28x. It is involved in determining if the results of certain operations are 0. It is also used for conditional operations. |
|----|---|
| Ν | Negative flag. N is new on the C28x. It is involved in determining if the results of certain operations are negative. It is also used for conditional operations. |
| V | Overflow flag. V has changed names from OV on the C2xLP. It flags overflow conditions in the accumulator. |
| РМ | Product shift mode. The PM has increased to a 3-bit register with additional capabilities. Below is a comparison of the PM register in the C2xLP and the C28x. Note that the register behaves differently depending on the operational mode of the C28x device. The XSPM instructions correspond to equivalent C2xLP instructions conversion. On the C2xLP, the PM bits corresponded to no shift at reset. On C28x, however, the PM corresponds to a left shift of 1 at reset. |

Table C–2. C2xLP Product Mode Shifter

| Bits | Shift Value | Instruction |
|------|---------------|-------------|
| 00 | no shift | SPM 0 |
| 01 | shift left 1 | SPM 1 |
| 10 | shift left 4 | SPM 2 |
| 11 | shift right 6 | SPM 3 |

| Table C–3. C28x Product Mode Sh |
|---------------------------------|
|---------------------------------|

| | C2xLP Source-Compatible Mode AMODE == 1 OBJMODE = 1 PAGE0 == 0 | | C28x Mode AMODE == 0 OBJMODE = 1 PAGE0 == 0 | | |
|------|---|-------------------|--|-------------|--|
| Bits | Shift Value | Instruction | Shift Value | Instruction | |
| 000 | shift left 1 | SPM +1 (or SPM 1) | shift left 1 | SPM +1 | |
| 001 | no shift | SPM 0 (or SPM 0) | no shift | SPM 0 | |
| 010 | shift right 1 | SPM –1 | shift right 1 | SPM –1 | |
| 011 | shift right 2 | SPM –2 | shift right 2 | SPM -2 | |
| 100 | shift right 3 | SPM –3 | shift right 3 | SPM –3 | |
| 101 | shift left 4 | SPM +4 (or SPM 2) | shift right 4 | SPM -4 | |
| 110 | shift right 5 | SPM –5 | shift right 5 | SPM –5 | |
| 111 | shift right 6 | SPM –6 (or SPM 3) | shift right 6 | SPM –6 | |

| OVC: | Overflow counter. OVC is new on the C28x. It can be viewed as an extension of the accumulator. For signed operations, the OVC |
|------|---|
| | counter is an extension of the overflow mode. For unsigned opera- tions, the OVC counter (OVCU) is an extension of the carry mode. |

- **DBGM: Debug enable mask bit.** DBGM is new on the C28x. It is analogous to the INTM bit and works in cooperation with the DBGIER register to globally enable interrupts in real-time emulation.
- PAGE0 PAGE0 addressing mode configuration bit. The PAGE0 bit is new on the C28x. It is used for compatibility to the C27x and should be left as 0 for users moving from the C2xLP to C28x.
- VMAP Vector map bit. The VMAP bit is new on the C28x. It determines from where in memory interrupt vectors will be fetched.
- **SPA Stack pointer alignment bit.** The SPA bit is new on the C28x. It is a flag used to determine if aligning the stack pointer caused an adjustment in the stack pointer address.
- LOOP Loop instruction status bit. The LOOP bit is new on the C28x. It is used in conjunction with the LOOPZ/LOOPNZ instructions.
- EALLOW Emulation access enable bit. The EALLOW bit is new on the C28x. It allows access to the emulation register on the C28x.
- **IDLE STAT IDLE status bit.** The IDLESTAT bit is new on the C28x. It flags an IDLE condition on the C28x, and is mainly used when returning from an interrupt.
- AMODE Address mode bit. The AMODE bit is new on the C28x. This mode bit is used to select between C28x addressing mode (AMODE == 0) and C2xLP addressing mode (AMODE == 1).
- **OBJMODE Object mode bit.** The OBJMODE bit is new on the C28x. It is used to select between C27x object mode (OBJMODE == 0) and C28x object mode (OBJMODE == 1). For users moving from C2xLP to C28x, this bit should always be set to 1.
 - **Note:** Upon reset of the C28x, this bit is set to 0 and needs to be changed in firmware.
- **MOM1MAP MO M1 map bit.** The MOM1MAP bit is new on the C28x. It is only used for C27x compatibility. For users transitioning from the C2xLP to C28x this bit should always be set to 1.
- **XF XF pin status bit.** The XF pin has the same function as on the C2xLP. Please note that the reset state has changed on the C28x.
- ARP Auxiliary register pointer. The ARP has the same functionality as on the C2xLP. It should, however, only be used when transitioning code to the C28x. The C28x has enhanced addressing modes which eliminate the need to keep track of the ARP.

The functionality of the remaining bits is the same on C28x as they are on C2xLP. It should be noted that although the functionality did not change, the bit position in the registers did. These bits are:

- □ Sign extension mode (SXM)
- Overflow mode (OVM)
- □ Test/control flag (TC)
- Carry bit (C)
- □ Interrupt global mask bit (INTM)

C.2.4 Register Reset Conditions

The reset conditions of internal registers have changed between the C2xLP and C28x as shown in Table C-4. Most C28x registers are cleared on a reset.

Differences in Table C–5 are highlighted in **bold**.

| C2xLP Register | C2xLP Reset | C28x Register | C28x Reset |
|-----------------|---------------|---------------|------------|
| Т | Х | XT | 0x0000000 |
| Р | Х | Р | 0x0000000 |
| ACC | Х | ACC | 0x0000000 |
| AR0-AR7 | Х | XAR0-XAR7 | 0x0000000 |
| PC | 0x0000 | PC | 0x3FFFC0 |
| ST0 | See Table C-5 | ST0 | 0x0000 |
| ST1 | See Table C-5 | ST1 | 0x080B |
| DP | Х | DP | 0x0000 |
| - | - | SP | 0x0400 |
| IMR | 0x00 | IER | 0x0000 |
| - | _ | DBGIER | 0x0000 |
| IFR | 0x0000 | IFR | 0x0000 |
| GREG | 0x0000 | - | - |
| - | - | RPC | 0x000000 |
| X = Uninitiated | | • | |

Table C–4. Reset Conditions of Internal Registers
| Reg | C2xLP Bit Name | C2xLP Reset Value | C28x Bit Name | C28x Reset Value |
|-----|-------------------|-------------------|--------------------------|--------------------|
| ST0 | DP | XXXXXXXXX | SXM | 0 |
| | INTM | 1 | OVM | 0 |
| | OVM | Х | TC | 0 |
| | OV | 0 | C | 0 |
| | ARP | XXX | Z | 0 |
| | | | Ν | 0 |
| | | | V | 0 |
| | | | PM | 000 (left shift 1) |
| ST1 | РМ | 00 (no shift) | INTM | 1 |
| | XF | 1 | DBGM | 1 |
| | С | 1 | PAGE0 | 0 |
| | SXM | 1 | VMAP | 1 |
| | TC | Х | SPA | 0 |
| | CNF | 0 | LOOP | 0 |
| | ARB | XXX | EALLOW | 0 |
| | | | IDLESTAT | 0 |
| | | | AMODE | 0 |
| | | | OBJMODE | 0 |
| | | | CNF not implement- ed | 0 |
| | | | M0M1MAP | 1 |
| | | | XF | 0 |
| | | | ARP | 000 |

Table C-5. Status Register Bits

C.3 Memory Map

The major changes between the C2xLP and C28x memory maps are outlined in this section. There are several differences between the C2xLP and C28x memory maps. These improvements are due to the expanded architecture of the C28x. The C28x CPU memory map ranges from 4G to 4M in data and program memory, respectively. However, C28x CPU-based devices may not use the entire memory range. See the device data sheet for the specific memory range applicable to that device.

Vectors. On the C2xLP, only one vector table is present at address 0x0000. These vectors were generally branch instructions to different interrupt service routines. On the C28x, the vector table can be placed in two different locations depending on the state of the VMAP input pin. On devices that do not pin out the VMAP signal, it is tied internal to the device. Generally, vectors will be located in non-volatile memory at 0x3FFFC0–0x3FFFFF. To take advantage of relocatable vectors or fetching vectors from fast internal memory space, place the vectors at address 0x00000–0x00003F. Often the C28x CPU interrupt vectors are expanded using external hardware logic. In such cases, see the related documents for the expanded vector map.

Memory space. On the C2xLP, the memory space for program, data, and I/O space is each 64K words. On the C28x, the program memory space is 4M words (22 address signals). The data memory space is 4G words (32 address signals). The global space (32K) and I/O space (64K) is generally used for C2xLP compatibility.

Program space. On the C2xLP CPU, program space could be mapped anywhere from (0x0–0xFFFF). With the extended address reach of the C28x (22 bits), the compatible region in program space for the C2xLP is 0x3F0000–0x3FFFF. Thus, any program memory on the C2xLP must be remapped to this upper region on the C28x. When the processor accesses program memory, the upper bits (bits 16–22) will be forced to all 1's when C2xLPcompatible instructions are used (See Appendix E).



Figure C-4. Memory Map Comparison (See Note A)

Note A: Memory map is not to scale.

C2xLP and C28x Architectural Differences C-13

Data memory. The C2xLP has three internal memory regions (B0, B1, B2) totaling 544 words. The C28x has two internal memory regions (M0,M1) totaling 1K words each. Note that for strict C2xLP compatibility, the memory regions are placed at the same addresses as noted in Table C–6.

Table C-6. B0 Memory Map

| C28x in C2xLP-Compatible Mode | C2xLP | |
|--|-----------------------------------|--|
| CNF Not Available | CNF = 0 | |
| B0 range mapped in M0 block 200 – 2FFh. | B0 in Data space | |
| (No mirroring of the block) | 100 – 1FFh (mirrored locations) | |
| | 200 – 2FFh | |
| CNF Not Available | CNF = 1 | |
| B0 range cannot be enabled in C2xLP-equivalent | B0 in program space | |
| program memory | FE00 – FEFFh (mirrored locations) | |
| | FF00 – FFFFh | |

I/O space. I/O space has remained on the C28x for compatibility reasons, and can only be accessed using IN and OUT/UOUT instructions. Not all C28x devices will support I/O space. See the data sheet of your particular device for details.

Global space. Global space is not supported on all C28x devices. See the data sheet specific to your device for details.

Reserved memory. Reserved memory regions have changed on the C28x. No user-defined memory or peripherals are allowed at addresses 0x800–0x9FF on the C28x. While using C2xLP-compatible mode, these addresses are reserved. It is recommended that C2xLP memory or peripherals be relocated to avoid memory conflicts.

Stack space. The C28x has a dedicated software stack pointer. This pointer is initialized to address 0x0400 (the beginning of block M1) at reset, and it grows upward in address. It is up to the user to move this stack pointer if needed in firmware.

Appendix D

C2xLP Migration Guidelines

The C28x DSP is source-code compatible with C2xLP DSP based devices. The C28x DSP assembler accepts all C2xLP mnemonics with the exception of a few instructions. This chapter provides guidelines for C2xLP code migration to a C28x device. C2xLP refers to the CPU used in all TMS320C24x, TMS320C24xx, and TMS320C20x DSP devices.

| Topi | C | Page |
|------|--|-------|
| D.1 | Introduction | . D-2 |
| D.2 | Recommended Migration Flow | . D-3 |
| D.3 | Mixing C2xLP and C28x Assembly | . D-6 |
| D.4 | Code Examples | . D-7 |
| D.5 | Reference Tables for Code Migration Topics | D-10 |

D.1 Introduction

This chapter provides guidelines that are intended for conversion from C2xLP assembly source to C28x object code. The conversion steps highlight the architectural changes between C2xLP and C28x operating modes. Future releases of documents will contain code conversion examples and software library modules facilitating the conversion from C2xLP mixed C and assembly source to C28x object code.

This chapter will be best understood if the reader has prior knowledge of Appendix C and Appendix E, as they explain the architectural and instructional enhancements between the C2xLP and C28x DSPs.

D.2 Recommended Migration Flow

Use the following steps (shown in Figure D-1) to migrate code:

- 1) Install the latest development tools for the C28x DSP (e.g. Code Composer Studio[™] version 2.x or higher)
- 2) Build the project with following C28x assembler options:

| -m20 | ; | enable C2xLP instructions |
|------|---|---|
| – g | ; | enable source level debug to view the $\ensuremath{\mathtt{C2xLP}}$ |
| | ; | instructions |
| -mw | ; | enable additional assembly checks |

Code Composer Studio 2.x will assemble all C2xLP instructions and map all the compatible instructions to their equivalent C28x instructions and mnemonics. Code Composer Studio 2.x disassembly will display the instructions in the memory as C28x mnemonics only. If the source is built with –g option, the relevant C2xLP source file will be also displayed and will facilitate C2xLP instruction readability during debug.

3) Memory map:

Define your C28x device memory map with C2xLP compatible memory sections. Build a linker command file (*.cmd). See Table D–8.

Select a C2xLP assembly source code *.asm for migration to C28x architecture.

4) Boot Code:

Add the C2xLP mode conversion code segment shown in section D.4.1 as the first set of instructions after reset.

After reset, the C28x powers up in C27x object–compatible mode. Adding these few lines of initialization code will place the device in the proper operating mode for executing reassembled C2xLP code.

Note: The C27x object-compatible mode is for use only for migration from the C27x CPU. It is a reserved operating mode for all C28x and C2xLP applications.

 This step will facilitate faster code conversion. In the C2xLP source file modify the interrupt section with suggestions from the reference table in section D.5.

In particular, modify the following types of code:

- a) IMR and IFR See the example code in section D.4.2.
- b) Context Save/Restore See the example code in section D.4.3
- c) Comment all the known incompatible instructions or map with equivalent instructions. See Table E-2 in Appendix E.





Legend: * represents user filename

- 6) Link the assembled code with the linker command file generated in Step 2. Relink if necessary to avoid any linker related errors.
- 7) Assemble or reassemble using the C28x assembler until the assembly is successful with no errors. The tables in section D.5 will help to resolve most of the errors during the assembly process. This will prepare a ***.obj** file, ready for C28x Linker processing.
- 8) The Linker output COFF file, ***.out**, will be the migrated code and should be ready for Debug and integration.

D.3 Mixing C2xLP and C28x Assembly

At this point your original C2xLP code will be running on the C28x device. To facilitate further migration to C28x code, there are special assembler directives that will facilitate mixing of C2xLP code and C28x code segments.

The .c28_amode and .lp_amode directives tell the assembler to override the assembler mode.

- .c28_amode The .c28_amode directive tells the assembler to operate in the C28x object mode (-v28).
- *.lp_amode* The .lp_amode directive tells the assembler to operate in C28x object accept C2xLP syntax mode (–m20).

These directives can be repeated throughout a source file.

For example, if a file is assembled with the -m20 option, the assembler begins the assembly in the C28x object - accept C2xLP syntax mode. When it encounters the .c28_amode directive, it changes the mode to C28x object mode and remains in that mode until it encounters an .lp_amode directive or the end of file.

Example In this example, C28x code is inserted in the existing C2xLP code.

```
; C2xLP source code
.lp amode
LDP
     #VarA
LACL
     VarA
LAR
      AR0 *+, AR2
SACL
      *+
CALL
      FuncA
; The C2xLP code in function FuncA is replaced with C28x Code
; using C28x addressing (AMODE = 0)
.c28 amode ; Override the assembler mode to C28x syntax
FuncA:
      C28ADDR
                          ; Set AMODE to 0 C28x addressing
      MOV DP, #VarB
      MOV AL, @VarB
      MOVL XAR0, *XAR0++
      MOV *XAR2++, AL
                         ; Change back the assembler mode to C2xLP.
      .lp amode
      LPADDR
                         ; Set AMODE to 1 to resume C2xLP addressing.
      LRET
```

D.4 Code Examples

D.4.1 Boot Code for C28x operating mode initalization

Note: The following code fragment must be placed in your code just after reset. This code will place the device in the proper operating mode to execute C2xLP converted code:

| Code | | Explanation | | |
|------|---------|--|--|--|
| SETC | OBJMODE | ;C28OBJ = 1 enable 28x object mode | | |
| CLRC | PAGE0 | ;PAGE0 = 0 not relevant for 28x mode, ;cleared to zero | | |
| SETC | AMODE | ;AMODE = 1 enable C2xLP compatible ;addressing mode | | |
| SETC | SXM | ;SXM = 1 for C2xLP at reset, SXM = 0 ;for 28x at reset | | |
| SETC | С | ;Carry bit =1 for C2xLP at reset, ;Carry bit = 0 for 28x at reset | | |
| SPM | 0 | ;Set product shift mode zero, that is PM bits = 001 compatible to ;C2xLP PM reset;mode | | |

D.4.2 IER/IFR Code

Table D–1. Code to Save Contents Of IMR (IER) And Disabling Lower Priority Interrupts At Beginning Of ISR

| C2xLP | | | C28x |
|-------|------|------------|--|
| INTx: | • | | INTx: . |
| | MAR | *,AR1 | AND IER, #~INT_MASK |
| | LDP | # O | |
| | LACL | IMR | |
| | SACL | *+ | Note: C28x saves IER as part of auto- |
| | AND | #~INT_MASK | matic context save operation and dis- |
| | SACL | IMR | ables the current interrupt automati- |
| | | | cally to prevent recursive interrupts. |
| | | | |

Table D-2. Code to Disable an Interrupt

| C2xLP | | | C28x |
|-------|------|--------|----------------------------|
| SETC | INTM | | AND IER, #~INTx |
| | LDP | # O | |
| | LACL | IMR | ;operation is atomic and |
| | AND | #~INTx | ; will not be interrupted. |
| | SACL | IMR | , |
| | CLRC | INTM | |

Table D-3. Code to Enable an Interrupt

| C2xLP | | | C28x |
|-------|------|-------|---------------------------|
| SETC | INTM | | |
| | LDP | # O | OR IER, #INTx |
| | LACL | IMR | |
| | OR | #INTx | ;operation is atomic and |
| | SACL | IMR | ;will not be interrupted. |
| CLRC | INTM | | |

Table D-4. Code to Clear the IFR Register

| C2xLP | | | C28x |
|--------|--------|-------------|--------------------------|
| ;write | e 1 to | clear | ;write 0 to clear |
| | SETC | INTM | AND IFR, #~INTx |
| | LDP | #0 | |
| | SPLK | #0FFFFh,IFR | ;operation is atomic and |
| CLRC | INTM | | ;will not be interrupted |

D.4.3 Context Save/Restore

The C28x automatically saves a number of registers on each interrupt. To perform a full context save, some additional code must be added. Table D–5 shows a typical full context save and restore for both processors.

| C2xLP Full Con | text Save/Restore | C28x Full Context Save/Restore | |
|----------------|----------------------|--|--|
| | | ;C28x automatically saves the | |
| | | following registers: | |
| INTx_ISR: | | ; T, STO, AH, AL, PH, PL, AR1, AR0, DP, ST1, | |
| ; context save | | ;DBGSTAT,IER,PC | |
| MAR *, AR1 | | | |
| MAR | *+ | INTx ISR: | |
| SST | #1,*+ | ;interrupt context save | |
| SST | #0,*+ | PUSH AR1H:AR0H ; 32-bit | |
| SACH | *+ | PUSH XAR2 ; 32-bit | |
| SACL | *+ | PUSH XAR3 ; 32-bit | |
| SPH | *+ | PUSH XAR4 ; 32-bit | |
| SPL | *+ | PUSH XAR5 ; 32-bit | |
| MPY | #1 | PUSH XAR6 ; 32-bit | |
| SPL | *+ | PUSH XAR7 ; 32-bit | |
| SAR | AR0, *+ | PUSH XT ; 32-bit | |
| SAR | AR2, *+ | | |
| SAR | AR3, *+ | | |
| SAR | AR4, *+ | ;interrupt code goes here | |
| SAR | AR5, *+ | | |
| SAR | AR6, *+ | | |
| SAR | AR7, *+ | ; interrupt context restore | |
| | | POP XT | |
| ;inter | rrupt code goes here | POP XAR7 | |
| | | POP XAR6 | |
| | | POP XAR5 | |
| ; context re | store | POP XAR4 | |
| MAR | *, AR1 | POP XAR3 | |
| MAR | *_ | POP XAR2 | |
| LAR | AR7, *- | POP AR1H:AR0H | |
| LAR | AR6, *- | IRET | |
| LAR | AR5, *- | | |
| LAR | AR4, *- | | |
| LAR | AR3, *- | | |
| LAR | AR2, *- | | |
| LAR | AR0, *- | | |
| SETC | INTM | | |
| MAR | *_ | | |
| SPM | 0 | | |
| LT | *+ | | |
| MPY | #1 | | |
| LT | *_ | | |
| MAR | *_ | | |
| LPH | *_ | | |
| LACL | *_ | | |
| ADD | *-, 16 | | |
| LST | #U, *- | | |
| LST | #⊥, *- | | |
| CLRC | | | |
| KE.I. | | | |
| | | | |

Table D–5. Full Context Save/Restore Comparison

D.5 Reference Tables for C2xLP Code Migration Topics

Table D–6 through Table D–10 explain the major differences between the C2xLP and C28x architectures and in their respective code generation process. These tables are organized to highlight the differences in interrupts, CPU registers, memory maps, instructions, registers, and syntax. While migrating the C2xLP code, check the tables for these key differences to make the necessary changes to the source to avoid assembler or linker errors.

Table D-6. C2xLP and C28x Differences in Interrupts

| | Migration topic | C2xLP | C28x |
|---|-----------------------------|---|---|
| 1 | Interrupt flag register | IFR – Memory mapped register | IFR is a CPU register |
| | | Write 1 to clear bits set in IFR | Write 0 to clear bits set in IFR |
| 2 | Interrupt enable register | IMR – Memory mapped register | Renamed as IER and is a CPU regis- ter |
| 3 | TRAP instruction | Only one TRAP vector | multiple,32– TRAP vectors |
| | | TRAP | TRAP 0, TRAP31 |
| | | Affects: INTM bit is not affected | Affects: INTM bit is set to 1 |
| 4 | INTR instruction syntax | INTR0 | INTR INTO |
| | | | |
| | | INTR31 | INTR INT31 |
| | | Affects: IFR not cleared | Affects: IFR cleared |
| | | IMR not affected | IER affected |
| | | INTM bit =1 | INTM bit =1 |
| 5 | NMI Instruction | NMI | TRAP NMI |
| 6 | CLRC INTM instruction | CLRC INTM instruction blocks all interrupts until the next in- | Interrupts enabled after the instruc- tion |
| | | | CLRC INTM |
| | | next_instn ;interrupts ;blocked | |
| | | ;until this ;executed | |
| 7 | Interrupt enable and return | CLRC INTM | IRET |
| | from interrupt service | RET | |

| | Migration topic | C2xLP | C28x |
|----|--|---|--|
| 8 | Interrupt enable and return from function call | CLRC INTM next_instn | next_instn CLRC INTM |
| 9 | Interrupts Vector | Uses Branch statements at the vector address. Ex: B Start ; assembly ; code ; opcode in memory 0x7980 ; branch ; instruction 0x0040 ; branch ; address | 32-bit absolute addresses. ; code in vector location 0x0040 (low address) 0x003F (high address) |
| 10 | Context save | No automatic context save See section D.3 for a full context save/restore example | Automatic context save of CPU regis- ters T, ST0, AH, AL, PH, PL, AR1, AR0, DP, ST1, DBGSTAT, IER, PC See Table D–5 for a full context save/ restore example |

Table D-6. C2xLP and C28x Differences in Interrupts (Continued)

| Table D-7. | C2xLP | and C28x | <i>Differences</i> | in S | Status | Reaisters |
|------------|-------|----------|--------------------|------|--------|-----------|
| | | | | | | |

| | Migration topic | C2xLP | C28x |
|---|--------------------------|---|---|
| 1 | Saving ST0/ST1 registers | Save: SST #0,mem ;store ST0 SST #1,mem ;store ST1 Restore: LST #0,mem ;load ST0 LST #1,mem ;load ST1 | Save: PUSH ST ;store ST0 to stack PUSH ST ;store ST1 to stack Restore: POP ST1 ;load ST1 ;from stack POP ST0 ;load ST0 ;from stack |
| 2 | ST0/ST1 bit differences | ST0/ST1 bits have CPU registers and status bits | ST0/ST1 bits are rearranged compared to C2xLP registers. |

| 3 | INTM bit in ST0 | Cannot be saved if ST0 register is saved | Saved along with ST0 register |
|---|------------------------------|--|---|
| 4 | Data page pointer DP save | DP save/restored along with STO. SST #0,mem ;store STO LST #0,mem ;load STO | <pre>DP is a register, hence explicit store/ restore is required. PUSH DP ; store DP ; to stack PUSH DP:ST1 ; 32-bit ; save POP DP ;load DP from ;stack POP DP:ST1 ; 32-bit ; restore</pre> |

Table D-7. C2xLP and C28x Differences in Status Registers (Continued)

| Table D–8. (| C2xLp and | C28x | Differences | in | Memory | Maps |
|--------------|-----------|------|-------------|----|--------|------|
|--------------|-----------|------|-------------|----|--------|------|

| | Migration topic | C2xLP | C28x |
|---|-----------------|-----------------------|-----------------------------|
| 1 | Program memory | 16-bit address | 22 – bit address |
| | | Size : 64kx16 | Size : 64kx16 |
| | | Range :0x0000-0xFFFFh | mapped to Range : |
| | | | 0x3F 0000h – 0x3F FFFFh |
| 2 | Data memory | Size : 64kx16 | Size : 64kx16 |
| | | Range :0x0000-0xFFFFh | mapped to Range : |
| | | | 0x00 0000h – 0x00 FFFFh |
| 6 | B2 Block | Size: 32 words | Located in M0 Block 1Kx16 |
| | | Range: 0x0060-0x007F | Size: 1K words |
| | | | Range: |
| | | | 0x00 0060 –0x00 07Fh |
| 7 | B1 Block | Size: 256 words | Located in M0 Block – 1Kx16 |
| | | Range: 0x0100-0x01FF | Not Mirrored |
| | | (mirrored) | Range: |
| | | : 0x0200-0x02FF | 0x00 0200 –0x00 02FFh |
| 8 | B0 Block | Mirrored locations | Located in M0 Block – 1Kx16 |
| | | Size: 256 words | Not Mirrored |
| | | Range: 0x0300-0x03FF | Range: |
| | | : 0x0400-0x04FF | 0x00 0300 –0x00 03FFh |

| | Migration topic | C2xLP | C28x |
|----|--------------------------------|---|---|
| 9 | CNF bit mapping of B0 Block | CNF bit maps B0 in data and program memory | Not applicable |
| | | CNF =0 – B0 in data memory | |
| | | Range: 0x0300-0x03FF | |
| | | : 0x0400-0x04FF | |
| | | CNF =1 - B0 in program memory | |
| | | Range: 0xFE00-0xFEFF | |
| | | : 0xFF00-0xFFFF | |
| 10 | Vector table range | Size: 32x16 words | Size 32x32 words |
| | | Range: 0x0000-0x003F | 0x3F FFC0 – 0x3F FFFF – at reset |
| | | | In C28x based DSP devices may use additional expanded vector table (e.g., PIE) |
| 11 | Internal SARAM mapping | Mapped as internal memory map | Reserved for emulation registers |
| | in data memory | | Range : 0x0800 -0x1000h |
| | | | |
| 5 | I/O space | Range : 0x0000 -0xFFFFh | Range : 0x0x00 000 –0x00 FFFFh |
| | | | I/O Space may or may not be imple- mented on a particular device. See the device datasheet for details. |
| 6 | Global space | Range : 0x8000 -0xFFFFh | Implemented via the XINTF |
| | | | Global Space may or may not be im- plemented on a specific C28x device. See the device datasheet for details. |

Table D-8. C2xLp and C28x Differences in Memory Maps (Continued)

| Table D-9. | C2xLP and | C28x Differences | s in Instructions | and Registers |
|------------|-----------|------------------|-------------------|---------------|
|------------|-----------|------------------|-------------------|---------------|

| | Migration topic | C2xLP | C28x |
|---|--|---|---|
| 1 | Conditional Instructions Branches, Calls, Returns | Can take more than one condi- tion in these instructions | The C28x assembler will automatically break the instructions into multiple instructions. |
| 2 | When are CPU Flags up- dated? | Conditional flags update on Ac- cumulator operation only | Conditional flags update on Accumu- lator, register and memory operations |
| 3 | Repeat instructions | Many instructions are repeatable | Same instructions are repeatable. For additional repeatable instructions see Table E-3. |

| | Migration topic | C2xLP | C28x |
|---|----------------------------------|--|---|
| 4 | GREG register | Memory mapped register | Memory mapped register in XINTF Global Space may or may not be im- plemented on a particular device. See the device data sheet for details. |
| 5 | ARx registers | ARx registers are 16-bit only LAR AR1, #0FFFFh ADRK #1 Result: AR1 = 0x0000h | XARn registers are 32 bits. Some in- structions access only the lower 16 bits known as ARn MOV XAR1, #0FFFFh ADD XAR1, #1 Result: XAR1 = 0x10000h |
| 6 | 2s complement subtraction to ARx | LAR AR1, #0FFFFh ADRK #0FE Result: AR1 = 0xFFFDh | MOV XAR1, #0FFFFh ADD XAR1,#0FE Result: XAR1 = 0x1FFFDh |
| 7 | I/O instructions | Supports IN, OUT instructions | Supports IN, OUT,UOUT I/O Space may or may not be imple- mented on a particular device. See the device datasheet for details. |
| 8 | Stack | Uses 8-deep Hardware stack C2xLP Compiler uses AR1 as Stack Pointer | Uses software stack pointer register (SP) Compiler will use SP register, as stack pointer |
| 9 | Program counter | 16 bits in size B 5000h ; Branch to 5000 ; address | 22 bits in size The C28x assembler will use special C2xLP compatible instructions that force the upper program address lines to 0x3F thus creating a 16-bit C2xLP compatible PC. B 0x3F5000 ; or XB 5000h |

Table D–9. C2xLP and C28x Differences in Instructions and Registers (Continued)

| | Migration topic | C2xLP | C28x |
|---|---|---|---|
| 1 | Mnemonic | Source or destination not always specified. LACL, source SACL, destination | Instructions are always of the form mnemonic destination, source MOV destination,source |
| 2 | Direct addressing syntax –@ symbol | LACL dma | MOV ACC, @@dma ; C2xLP mode MOV ACC, @dma ; 28x mode @@ – means 128 word data page @ – means 64 word data page |
| 3 | Indirect address pointer buffer, ARB | In indirect addressing, Auxiliary register will be pointed by ARP register in ST0. ARB is ARP pointer buffer in ST1. MAR *,AR2 ; ARP =AR2 LACL * | No ARB equivalent in 28x. Selected ARx is referenced in the in- struction itself. MOV ACC,*AR2 |
| 4 | New Address pointers syntax – *(0 | BLDD #4545h,RegA | MOV @REGA, *(0:0x4545) |
| 5 | Repeat instructions syntax change – | No additional syntax RPT #5 NOP | Uses syntax with repeat instruc- tions RPT #5 NOP |
| 6 | Reserved register names Application code should not use these reserved words | ST0, ST1, IFR, IMR, GREG | ST0, ST1, AH, AL, PH, PL,T, TL, XAR0, XAR1, XAR2, XAR3, XAR4, XAR5, XAR6, XAR7, DP, ST1, DBGSTAT, IER, PC, RPC |
| 7 | Increment/Decrement syntax change | MAR *,AR2 LACL *+ LACL *- | MOV ACC, *AR2++ MOV ACC, *AR2 |
| 8 | Sniπ syntax change | LAGE dma, 4 | MUV ACC, dma <<4 |

Table D-10. Code Generation Tools and Syntax Differences

| | Migration topic | C2xLP | C28x |
|----|---|---|--|
| 9 | Number radix usage | x .set 09 ;Assembler ;accepts ;this as ;decimal 9 | x .set 9 Avoid leading zeros, else the assem- bler will be use this as octal number. |
| 10 | Order of precedence in ex- pressions – Syntax change | Expressions in assembly state- ments do not require parenthesis. x .set A< <b =="" c="">>D | Expressions in assembly statements do require parenthesis. x .set (A< <b =="" c="">>D) |
| 11 | Tools Directives | .mmregs ; reserved register use .port .globl | not applicable not applicable .global |
| 12 | Macros | Useful in coding style | Useful in coding style All C2xLP Macros are not directly used Convert them individually to 28x mode. |
| 13 | Assembler options | -v2xx | -m20, -v28 |

Table D-10. Code Generation Tools and Syntax Differences (Continued)

Appendix E

C2xLP Instruction Set Compatibility

This appendix highlights the differences in syntax between the C2xLP and the C28x instructions, and details which C2xLP compatible instructions are repeatable on the C28x. The C28x assembler accepts both C28x and C2xLP assembly source syntax. This enables you to quickly port C2xLP code with minimal effort. Additionally, all compatible C2xLP instructions have an equivalent C28x style syntax. The C28x disassembler will show the C28x equivalent syntax.

| Topi | c | Page |
|------|--------------------------|------|
| E.1 | Condition Tests on Flags | E-2 |
| E.2 | C2xLP vs. C28x Mnemonics | E-3 |
| E.3 | Repeatable Instructions | E-9 |

E.1 Condition Tests on Flags

On the C28x, all EQ/NEQ/GT/LT/LEQ conditional tests are performed on the state of the Z and N flags. On the C2xLP, the same condition tests are performed on the contents of the ACC register.

| Table E-1. | C28x and | C2xLP Flags |
|------------|----------|-------------|
|------------|----------|-------------|

| Designation | C28x Modes | C2xLP Equivalent |
|-------------|---------------------------|------------------|
| NEQ | != 0 | ACC != 0 |
| EQ | == 0 | ACC == 0 |
| GT | > 0 | ACC > 0 |
| GEQ | >= 0 | ACC >= 0 |
| LT | < 0 | ACC < 0 |
| LEQ | <= 0 | ACC <= 0 |
| н | higher | - |
| HIS, C | higher or same, carry set | C == 1 |
| LO, NC | lower, carry clear | C == 0 |
| LOS | lower or same | - |
| NOV | no overflow | OV == 0 |
| OV | overflow | OV == 1 |
| NTC | TC == 0 | TC == 0 |
| тс | TC == 1 | TC == 1 |
| NBIO | test BIO input == 0 | BIO == 0 |
| UNC | unconditional | UNC |

On the C28x, the Z and N flags are set on all ACC operations. That includes ACC loads. Therefore, the Z and N flags reflect the current state of the ACC immediately after an operation on the ACC.

E.2 C2xLP vs. C28x Mnemonics

Table E–2 lists the C2xLP instructions with the C28x equivalent syntax. The C28x assembler will accept either the C2xLP syntax or the equivalent C28x syntax. The disassembler will decode and display the C28x syntax.

The C2xLP cycle count numbers shown are for zero wait-state internal memory, where n equals the number of repetitions (i.e., if an instruction is repeated, using the RPT instruction for repeatable instructions, n times it is executed n+1 times).

| | C2xLP | | | | C28x | | |
|------------------|--------------|--------|------|------------------|--|--------|------|
| Instruc- tion | Mnemonic | Cycles | Size | Instruc- tion | Mnemonic | Cycles | Size |
| ABS | | n+1 | 16 | ABS | ACC | 1 | 16 |
| ADD | loc16[,0] | n+1 | 16 | ADD | ACC,loc16 {<<0} | n+1 | 16 |
| ADD | loc16,115 | n+1 | 16 | ADD | ACC,loc16 << 115 | n+1 | 32 |
| ADD | loc16,16 | n+1 | 16 | ADD | ACC,loc16 << 16 | n+1 | 16 |
| ADD | #8bit | 1 | 16 | ADDB | ACC,#8bit | 1 | 16 |
| ADD | #16bit[,015] | 2 | 32 | ADD | ACC,#16bit {<<015} | 1 | 32 |
| ADDC | loc16 | n+1 | 16 | ADDCU | ACC,loc16 | 1 | 16 |
| ADDS | loc16 | n+1 | 16 | ADDU | ACC,loc16 | n+1 | 16 |
| ADDT | loc16 | n+1 | 16 | ADD | ACC,loc16 << T | n+1 | 32 |
| ADRK | #8bit | 1 | 16 | ADRK | #8bit | 1 | 16 |
| AND | loc16 | n+1 | 16 | AND | ACC,loc16 | n+1 | 16 |
| AND | #16bit,16 | 2 | 32 | AND | ACC,#16bit<<16 | 1 | 32 |
| AND | #16bit[,015] | 2 | 32 | AND | ACC,loc16 {<< 015} | 1 | 32 |
| APAC | | n+1 | 16 | ADDL | ACC, P< <pm< td=""><td>n+1</td><td>16</td></pm<> | n+1 | 16 |
| В | pma | 4 | 32 | ХВ | pma,UNC | 7 | 32 |
| В | pma,*,ARn | 4 | 32 | ХВ | pma,*,ARPn | 4 | 32 |
| В | pma,*ind | 4 | 32 | NOP XB | *ind pma, UNC | 8 | 32 |
| В | pma,*ind,ARn | 4 | 32 | NOP XB | *ind pma,*,ARPn | 5 | 48 |

Table E-2. C2xLP Instructions and C28x Equivalent Instructions

| | C2xLP | | | | C28x | | |
|------------------|----------------------------------|--------|------|----------------------------|---|--------|----------|
| Instruc- tion | Mnemonic | Cycles | Size | Instruc- tion | Mnemonic | Cycles | Size |
| BACC | | 4 | 16 | ХВ | *AL | 7 | 16 |
| BANZ | pma,*ind[,ARn] | 4/2 | 32 | XBANZ | <pre>pma,*ind[,ARAPn]</pre> | 4/2 | 32 |
| BANZ | pma,*BR0+/*BR0-[,ARn] | 4/2 | 32 | | Not applicable | | |
| BCND | pma[,COND] | 4/2 | 32 | XB or SB | pma,COND #8bitOff,COND | 7/4 | 32 16 |
| BCND | pma , COND1 , COND2 , , CONDn | 4/2 | 32 | SB SB XB skip: | skip,opposite of COND1 skip,opposite of COND2 pma,CONDn | 7+ | 48+ |
| BIT | loc16,15-bit | n+1 | 16 | TBIT | loc16,#bit | 1 | 16 |
| BITT | loc16 | n+1 | 16 | TBIT | loc16,T | 1 | 32 |
| BLDD | #src_addr,loc16 | n+3 | 32 | MOV | <pre>loc16,*(0:src_addr)</pre> | n+2 | 32 |
| BLDD | loc16,#dest_addr | n+3 | 32 | MOV | *(0:dest_addr),loc16 | n+2 | 32 |
| BLPD | <pre>#pma,loc16</pre> | n+3 | 32 | XPREAD | loc16,*(pma) | n+2 | 32 |
| CALA | | 4 | 16 | XCALL | *AL | 7 | 16 |
| CALL | pma | 4 | 32 | XCALL | pma,UNC | 7 | 32 |
| CALL | pma,*,ARn | 4 | 32 | XCALL | pma,*,ARPn | 4 | 32 |
| CALL | pma,*ind | 4 | 32 | NOP XCALL | *ind pma,UNC | 8 | 48 |
| CALL | pma,*ind,ARn | 4 | 32 | NOP XCALL | *ind pma,*,ARPn | 5 | 48 |
| CC | pma,COND | 4/2 | 32 | XCALL | pma,COND | 7/4 | 32 |
| СС | pma,COND1,,CONDn | 4/2 | 32 | SB SB XCALL skip: | skip,opposite of COND1 skip,opposite of COND2 pma,CONDn | 7+ | 48+ |
| CLRC | INTM | n+1 | 16 | | See Table D-6. | | |

Table E-2. C2xLP Instructions and C28x Equivalent Instructions (Continued)

| | C2xLP | | | | C28x | | |
|------------------|-----------------|--------|------|------------------|-------------------|--------|------|
| Instruc- tion | Mnemonic | Cycles | Size | Instruc- tion | Mnemonic | Cycles | Size |
| CLRC | XF/OVM/SXM/TC/C | n+1 | 16 | CLRC | XF/OVM/SXM/TC/C | 2,1 | 16 |
| CLRC | CNF | n+1 | 16 | | Not applicable | | |
| CMPL | | n+1 | 16 | NOT | ACC | 1 | 16 |
| CMPR | 0/1/2/3 | n+1 | 16 | CMPR | 0/1/2/3 | 1 | 16 |
| DMOV | loc16 | n+1 | 16 | DMOV | loc16 | n+1 | 16 |
| IDLE | | 1 | 16 | IDLE | | 5 | 16 |
| IN | loc16,PA | 2(n+1) | 32 | IN | loc16,*(PA) | n+2 | 32 |
| INTR | к | 4 | 16 | | Not applicable | | |
| LACC | loc16[,0] | n+1 | 16 | MOV | ACC,loc16 [<< 0] | 1 | 16 |
| LACC | loc16,115 | n+1 | 16 | MOV | ACC,loc16 << 115 | 1 | 32 |
| LACC | loc16,16 | n+1 | 16 | MOV | ACC,loc16 << 16 | 1 | 16 |
| LACC | #16bit,015 | 2 | 32 | MOV | ACC,#16bit << 015 | 1 | 32 |
| LACL | loc16 | n+1 | 16 | MOVU | ACC,loc16 | 1 | 16 |
| LACL | #8bit | 1 | 16 | MOVB | ACC,#8bit | 1 | 16 |
| LACT | loc16 | n+1 | 16 | MOV | ACC,loc16 << T | 1 | 32 |
| LAR | ARn,loc16 | 2(n+1) | 16 | MOVZ | ARn,loc16 | 1 | 16 |
| LAR | ARn,#8bit | 2 | 16 | MOVB | XARn,#8bit | 1 | 16 |
| LAR | ARn,#16bit | 2 | 32 | MOVL | XARn,#22bit | 1 | 32 |
| LDP | loc16 | 2(n+1) | 16 | | Not applicable | | |
| LDP | #9bit | 2 | 16 | MOVZ | DP,#10bit >> 1 | 1 | 16 |
| LPH | loc16 | n+l | 16 | MOV | PH,loc16 | 1 | 16 |
| LST | #0/1,loc16 | 2(n+1) | 16 | | See Table D-7 | | |
| LT | loc16 | n+l | 16 | MOV | T,loc16 | 1 | 16 |
| LTA | loc16 | n+l | 16 | MOVA | T,loc16 | n+1 | 16 |

Table E–2. C2xLP Instructions and C28x Equivalent Instructions (Continued)

| | C2xLP | | | | C28x | | |
|------------------|-----------------------|--------|------|------------------|--|--------|------|
| Instruc- tion | Mnemonic | Cycles | Size | Instruc- tion | Mnemonic | Cycles | Size |
| LTD | loc16 | n+l | 16 | MOVAD | T,loc16 | 1 | 16 |
| LTP | loc16 | n+l | 16 | MOVP | T,loc16 | 1 | 16 |
| LTS | loc16 | n+l | 16 | MOVS | T,loc16 | n+1 | 16 |
| MAC | pma,loc16 | n+3 | 32 | XMAC | P,loc16,*(pma) | n+2 | 32 |
| MACD | pma,loc16 | n+3 | 32 | XMACD | P,loc16,*(pma) | n+2 | 32 |
| MAR | <pre>*ind[,ARn]</pre> | n+l | 16 | NOP | <pre>*ind[,ARPn]</pre> | n+1 | 16 |
| MPY | loc16 | n+l | 16 | MPY | P,T,loc16 | 1 | 16 |
| MPY | #13bit | 1 | 16 | MPY | P,@T,#16bit | 1 | 32 |
| MPYA | loc16 | n+l | 16 | МРҮА | P,T,loc16 | n+1 | 16 |
| MPYS | loc16 | n+l | 16 | MPYS | P,T,loc16 | n+1 | 16 |
| MPYU | loc16 | n+l | 16 | MPYU | P,T,loc16 | 1 | 16 |
| NEG | | n+l | 16 | NEG | ACC | 1 | 16 |
| NMI | | 4 | 16 | | Not applicable | | |
| NOP | | n+l | 16 | NOP | | n+1 | 16 |
| NORM | */*+/*-/*0+/*0- | n+l | 16 | NORM | ACC,*/*++/*/*0++/*0 | n+4 | 16 |
| NORM | *BR0+/*BR0- | n+l | 16 | | Not applicable | | |
| OR | loc16 | n+l | 16 | OR | ACC,loc16 | n+1 | 16 |
| OR | #16bit,16 | 2 | 32 | OR | ACC,#16bit<<16 | 1 | 32 |
| OR | #16bit[,015] | 2 | 32 | OR | ACC,#16bit {<< 015} | 1 | 32 |
| OUT | loc16,PA | 3(n+1) | 32 | OUT | *(PA),loc16 | 4 | 32 |
| PAC | | n+l | 16 | MOV | ACC, P< <pm< td=""><td>1</td><td>16</td></pm<> | 1 | 16 |
| POP | | n+l | 16 | MOVU | ACC, *SP | 1 | 16 |
| POPD | loc16 | n+l | 16 | POP | loc16 | 2 | 16 |
| PSHD | loc16 | n+l | 16 | PUSH | loc16 | 2 | 16 |
| PUSH | | n+l | 16 | MOV | *SP++,AL | n+1 | 16 |

Table E-2. C2xLP Instructions and C28x Equivalent Instructions (Continued)

| C2xLP | | | C28x | | | | |
|------------------|----------------------|--------|------|------------------|--|--------|------|
| Instruc- tion | Mnemonic | Cycles | Size | Instruc- tion | Mnemonic | Cycles | Size |
| RET | | 4 | 16 | XRETC | UNC | 7 | 16 |
| RETC | COND | 4/2† | | XRETC | COND | 7/4 | 16 |
| RETC | COND1, COND2,, CONDn | 4/2 | 16 | SB SB | \$10,opposite of COND1 \$10,opposite of COND2 | 7+ | 48+ |
| | | | | XRETC \$10: | CONDn | | |
| ROL | | n+l | 16 | ROL | ACC | n+1 | 16 |
| ROR | | n+l | 16 | ROR | ACC | n+1 | 16 |
| RPT | loc16 | 1 | 16 | RPT | loc16 | 1 | 16 |
| RPT | #8bit | 1 | 16 | RPT | #8bit | 1 | 16 |
| SACH | loc16[,0] | n+l | 16 | MOV | loc16,AH | n+1 | 16 |
| SACH | loc16,1 | n+l | 16 | MOVH | loc16,ACC << 1 | n+1 | 16 |
| SACH | loc16,27 | n+l | 16 | MOVH | loc16,ACC << 27 | n+1 | 32 |
| SACL | loc16[,0] | n+l | 16 | MOV | loc16,AL | n+1 | 16 |
| SACL | loc16,1 | n+l | 16 | MOV | loc16,ACC << 1 | n+1 | 16 |
| SACL | loc16,27 | n+l | 16 | MOV | loc16,ACC << 27 | n+1 | 32 |
| SAR | ARn,loc16 | n+l | 16 | MOV | loc16,ARn | 1 | 16 |
| SBRK | #8bit | 1 | 16 | SBRK | #8bit | 1 | 16 |
| SETC | INTM | n+l | 16 | SETC | INTM | 2 | 16 |
| SETC | XF/OVM/SXM/TC/C | n+l | 16 | SETC | XF/OVM/SXM/TC/C | 2,1 | 16 |
| SETC | CNF | n+l | 16 | | Not applicable | | |
| SFL | | n+l | 16 | LSL | ACC,1 | n+1 | 16 |
| SFR | | n+l | 16 | SFR | ACC,1 | n+1 | 16 |
| SPAC | | n+l | 16 | SUB | ACC, P< <pm< td=""><td>n+1</td><td>16</td></pm<> | n+1 | 16 |
| SPH | loc16 | n+l | 16 | MOVH | loc16,P | n+1 | 16 |
| SPL | loc16 | n+l | 16 | MOV | loc16,P | n+1 | 16 |

Table E–2. C2xLP Instructions and C28x Equivalent Instructions (Continued)

| | C2xLP | | | | C28x | | |
|------------------|---------------|--------|------|------------------|---------------------|--------|------|
| Instruc- tion | Mnemonic | Cycles | Size | Instruc- tion | Mnemonic | Cycles | Size |
| SPLK | #0x0000,loc16 | 2 | 32 | MOV | loc16,#0 | n+1 | 16 |
| SPLK | #16bit,loc16 | 2 | 32 | MOV | loc16,#16bit | n+1 | 32 |
| SPM | 0 | 1 | 16 | SPM | 0 | 1 | 16 |
| SPM | 1 | 1 | 16 | SPM | 1 (or +1) | 1 | 16 |
| SPM | 2 | 1 | 16 | SPM | 2 (or +4) | 1 | 16 |
| SPM | 3 | 1 | 16 | SPM | 3 (or -6) | 1 | 16 |
| SQRA | loc16 | n+l | 16 | SQRA | loc16 | n+1 | 32 |
| SQRS | loc16 | n+l | 16 | SQRS | loc16 | n+1 | 32 |
| SST | #0/1,loc16 | n+l | 16 | | Not applicable | | |
| SUB | loc16[,0] | n+l | 16 | SUB | ACC,loc16 {<< 0} | n+1 | 16 |
| SUB | loc16,115 | n+l | 16 | SUB | ACC,loc16 << 115 | n+1 | 32 |
| SUB | loc16,16 | n+l | 16 | SUB | ACC,loc16 << 16 | n+1 | 16 |
| SUB | #8bit | 1 | 16 | SUBB | ACC,#8bit | 1 | 16 |
| SUB | #16bit[,015] | 2 | 32 | SUB | ACC,#16bit {<< 015} | 1 | 32 |
| SUBB | loc16 | n+l | 16 | SUBU | ACC,loc16 | 1 | 16 |
| SUBC | loc16 | n+l | 16 | SUBCU | ACC,loc16 | n+1 | 16 |
| SUBS | loc16 | n+l | 16 | SUBU | ACC,loc16 | n+1 | 16 |
| SUBT | loc16 | n+l | 16 | SUB | ACC,loc16 << T | n+1 | 32 |
| TBLR | loc16 | n+3 | 16 | XPREAD | loc16,*AL | n+4 | 32 |
| TBLW | loc16 | n+3 | 16 | XPWRITE | *AL,loc16 | n+4 | 32 |
| TRAP | | 4 | 16 | | Not applicable | | |
| XOR | loc16 | n+l | 16 | XOR | ACC,loc16 | n+1 | 16 |
| XOR | #16bit,16 | 2 | 32 | XOR | ACC,#16bit<<16 | 1 | 32 |
| XOR | #16bit[,015] | 2 | 32 | XOR | ACC,#16bit [<< 015] | 1 | 32 |
| ZALR | loc16 | n+l | 16 | ZALR | ACC,loc16 | 1 | 32 |

Table E-2. C2xLP Instructions and C28x Equivalent Instructions (Continued)

E.3 Repeatable Instructions

Not all of the repeatable instructions on the C2xLP are repeatable on the C28x. The ones that were not made repeatable do not make sense to repeat from a functionality standpoint. Also, some instructions that were not repeatable on the C2xLP are repeatable on the C28x.

Table E–3 shows which C2xLP operations are repeatable, and which ones are repeatable on the C28x.

| C2xLP Instruction | C2xLP Repeatable | C28x Repeatable |
|-------------------------------|---------------------|--------------------|
| ABS | Х | |
| ADD mem,shift1 | Х | Х |
| ADDC mem | Х | |
| ADDS mem | Х | Х |
| ADDT mem | Х | Х |
| AND mem | х | х |
| APAC | x | Х |
| BIT mem,bit_code | Х | |
| BITT mem | Х | |
| BLDD #addr,mem | Х | Х |
| BLDD mem,#addr | Х | Х |
| BLPD #pma,mem | Х | Х |
| CLRC CNF/XF/INTM/OVM/SXM/TC/C | Х | |
| CMPL | Х | |
| CMPR constant | Х | |
| DMOV mem | Х | Х |
| IN mem,PA | Х | Х |
| INTR K | Х | |
| LACC mem[,shift1] | Х | |
| LACL mem | Х | |

Table E–3. Repeatable Instructions for the C2xLP and C28x

C2xLP Instruction Set Compatibility E-9

| C2xLP Instruction | C2xLP Repeatable | C28x Repeatable |
|---------------------|---------------------|--------------------|
| LACT mem | Х | |
| LAR AR,mem | Х | |
| LDP mem | Х | |
| LPH mem | Х | |
| LST #n,mem | Х | |
| LT mem | Х | |
| LTA mem | Х | Х |
| LTD mem | Х | |
| LTP mem | Х | |
| LTS mem | Х | Х |
| MAC pma,mem | Х | Х |
| MACD pma,mem | Х | Х |
| MAR {ind}[,nextARP] | Х | Х |
| MPY mem | Х | |
| MPY #k | Х | |
| MPYA mem | Х | Х |
| MPYS mem | Х | Х |
| MPYU mem | Х | |
| NEG | Х | |
| NOP | Х | Х |
| NORM {ind} | Х | Х |
| OR mem | Х | Х |
| OUT mem,PA | Х | Х |
| PAC | Х | |
| POP | Х | |
| POPD mem | Х | |

Table E-3. Repeatable Instructions for the C2xLP and C28x (Continued)

| C2xLP Instruction | C2xLP Repeatable | C28x Repeatable |
|-------------------------------|---------------------|--------------------|
| PSHD mem | Х | |
| PUSH | Х | |
| ROL | Х | Х |
| ROR | Х | Х |
| SACH mem[,shift] | Х | Х |
| SACL mem[,shift] | Х | Х |
| SAR AR,mem | Х | |
| SETC CNF/XF/INTM/OVM/SXM/TC/C | Х | |
| SFL | Х | Х |
| SFR | Х | Х |
| SPAC | Х | Х |
| SPH mem | Х | Х |
| SPL mem | Х | Х |
| SPLK #lk,mem | Х | Х |
| SQRA mem | Х | Х |
| SQRS mem | Х | Х |
| SST #n,mem | Х | |
| SUB mem[,shift1] | Х | Х |
| SUBB mem | Х | |
| SUBC mem | Х | Х |
| SUBS mem | Х | Х |
| SUBT mem | Х | Х |
| TBLR mem | Х | Х |
| TBLW mem | Х | Х |
| XOR mem | Х | Х |
| ZALR mem | Х | |

Table E-3. Repeatable Instructions for the C2xLP and C28x (Continued)

C2xLP Instruction Set Compatibility E-11

Appendix F

Migration From C27x to C28x

This appendix highlights the architecture differences between the C27x and the C28x and describes how to migrate your code from a C27x-based design to a C28x-based design.

Topic

Page

| F.1 | Architecture Changes F-2 |
|-----|------------------------------------|
| F.2 | Moving to C28x Object F-9 |
| F.3 | Migrating to C28x Object Code F-11 |
| F.4 | Compiling C28x Source Code F-16 |

F.1 Architecture Changes

Certain changes to the architecture that are important when migrating from the C27x to the C28x include:

- Changes to registers
- Full context save and restore
- B0/B1 memory map consideration

F.1.1 Changes to Registers

The register modifications from the C27x are shown in Figure F–1. Shaded registers highlight the changes or enhancements for the C28x.

| Figure | F-1. | C28x | Registers |
|--------|------|------|-----------|
| | | | |

| T(16) | TL(16) | XT(32) | ST0(16) | IER(16) |
|--------|--------|---------|---------|------------|
| PH(16) | PL(16) | P(32) | ST1(16) | DBGIER(16) |
| AH(16) | AL(16) | ACC(32) | | IFR(16) |

| | | | | 1 |
|----------|-------|---------|------------------|----------|
| | | SP(| 16) | |
| | DP(| 16) | 6/7bit offset | |
| AR0H | H(16) | AR0(16) | | XAR0(32) |
| AR1H(16) | | AR1(16) | | XAR1(32) |
| AR2H | ł(16) | AR2(16) | | XAR2(32) |
| AR3F | ł(16) | AR3(16) | | XAR3(32) |
| AR4H(16) | | AR4(16) | | XAR4(32) |
| AR5H(16) | | AR5(16) | | XAR5(32) |
| AR6H | (16) | AR6 | (16) | XAR6(32) |
| AR7H | 1(16) | AR7 | (16) | XAR7(32) |
| | | PC(22) | | |
| | | RPC(22) | | |

A brief description of the register modifications is given below:

| XT(32), TL(16): | The T register is increased to 32-bits and called the XT register. The existing C27x T register |
|-----------------|---|
| | portion represents the upper 16-bits of the new 32-bit register. The additional 16-bits, called |
| | the TL portion, represents the lower 16-bits. |
| | |

- XAR0,..,XAR7(32): All of the AR registers are stretched to 32-bits. This enables a full 22-bit address. For addressing operations, only the lower 22-bits of the registers are used, the upper 10-bits are ignored. For operations between the ACC, all 32-bits are valid (register addressing mode @XARx). For 16-bit operations to the low 16-bit of the registers (register addressing mode @ARx), the upper 16-bits are ignored.
- RPC(22): This is the return PC register. When a call operation is performed, the return address is saved in the RPC register and the old value in the RPC is saved on the stack (in two 16-bit operations). When a return operation is performed, the return address is read from the RPC register and the value on the stack is written into the RPC register (in two 16-bit operations). The net result is that return operations are faster (4 instead of 8 cycles)
- SP(16): By default the C28x SP register is initialized to 0x400 after a reset.
- ST0 (16): Shaded items indicate a change or addition from the C27x

| Bit(s) | Mnemonic | Description | Reset Value | R/W |
|--------|----------|-------------------------|--------------|-----|
| 0 | SXM | Sign Extension Mode Bit | 0 | R/W |
| 1 | OVM | Overflow Mode Bit | 0 | R/W |
| 2 | тс | Test Control Bit | 0 | R/W |
| 3 | С | Carry Bit | 0 | R/W |
| 4 | Z | Zero Condition Bit | 0 | R/W |
| 5 | Ν | Negative Condition Bit | 0 | R/W |
| 6 | V | Overflow Condition Bit | 0 | R/W |
| 9:7 | PM | Product Shift Mode | 0 (+1 shift) | R/W |
| 15:10 | OVC/OVCU | ACC Overflow Counter | 0 | R/W |

Table F-1. ST0 Register Bits

PM:

Functionality of the Product Shift Mode changes if the AMODE bit in ST1 is set to 1. C27x users will not modify the AMODE bit and PM will function as they did on the C27x.

OVC/OVCU:

The overflow counter is modified so that it behaves differently for signed or unsigned operations. For signed operations (OVC), it behaves as it does on the C27x (increment for positive overflow, decrement for negative underflow of a signed number). For unsigned operations (OVCU), the overflow counter increments for an ADD operation when there is a carry generated and decrements for a SUB operation when a borrow is generated. Basically, in unsigned mode, the OVCU behaves like a carry (C) counter and in signed mode the OVC behaves like an overflow (V) counter.

| Bit(s) | Syntax | Description | Reset Value | R/W |
|--------|----------|---------------------------------|--------------------|-----|
| 0 | INTM | Interrupt Enable Mask Bit | 1 (disabled) | R/W |
| 1 | DBGM | DeBug Enable Mask Bit | 1 (disabled) | R/W |
| 2 | PAGE0 | PAGE0 Direct/Stack Address Mode | 0 | R/W |
| 3 | VMAP | Vector Map Bit | VMAP input | R/W |
| 4 | SPA | Stack Pointer Align Bit | 0 | R/W |
| 5 | LOOP | Loop Instruction Status Bit | 0 | R |
| 6 | EALLOW | Emulation Access Enable Bit | 0 | R/W |
| 7 | IDLESTAT | IDLE Status Flag Bit | 0 | R |
| 8 | AMODE | Address Mode Bit | 0 | R/W |
| 9 | OBJMODE | Object Compatibility Mode Bit | 0 | R/W |
| 10 | RESERVED | Reserved for future use | 0 | R |
| 11 | M0M1MAP | M0 and M1 Mapping Mode Bit | 1 | R |
| 12 | XF | XF Status Bit | 0 | R/W |
| 15:13 | ARP | Auxiliary Register Pointer | 0 | R/W |

Table F–2. ST1 Register Bits

AMODE: This mode selects the appropriate addressing mode decodes for compatibility with the C2xLP device. For all C27x/C28x based projects leave this bit as 0.

- OBJMODE: This mode is used to select between C27x object mode (OBJMODE == 0) and C28x object mode (OBJMODE == 1) compatibility. This bit is set by the "C28OBJ" (or "SETC OBJMODE") instructions. This bit is cleared by the "C27OBJ" (or "CLRC OBJMODE") instructions. The pipeline is flushed when setting or clearing this bit using the given instructions. This bit can be saved and restored by interrupts and when restoring the ST1 register. This bit is set to 0 on reset.
- M0M1MAP: This mode is used to remap block M0 and M1 in program memory space as discussed in detail in section F.1.2. This bit is set by the "C28MAP" (or "SETC M0M1MAP") instructions. This bit is cleared by the "C27MAP" (or "CLRC M0M1MAP") instructions. The pipeline is flushed when setting or clearing this bit using the given instructions. This bit cannot be restored by interrupts and when restoring the ST1 register (read only).
- XF: This bit reflects the current state of the XFS output signal. This signal is for C2xLP compatibility and is not used by C27x users.

F.1.2 Full Context Save and Restore

On both C27x and C28x, the registers in Figure F–2 are automatically saved on the stack on an interrupt or trap operation and automatically restored on an IRET instruction.

| 31 | 16 | 1 0 |
|---------|----|-----|
| Т | | ST0 |
| AH | | AL |
| PH | | PL |
| AR1 | | AR0 |
| DP | | ST1 |
| DBGSTAT | | IER |
| PCH | | PCL |

Figure F–2. Full Context Save/Restore

Due to the register changes described in section F.1.1. C28x additional registers must be saved for a full-context store. Figure F–3 shows the difference between a C27x and C28x full-context save/restore for an interrupt or trap.


| IntX: | ; 8 cycles |
|-------|-------------------|
| push | AR3:AR2 |
| | push AR5:AR |
| | push XAR6 |
| | push XAR7 |
| | ; + 4 = 12 cycles |
| | |
| | |
| | |
| | pop XAR7 |
| | pop XAR6 |
| | pop AR5:AR4 |
| | pop AR3:AR2 |
| | iret |
| | ; 12 cycles |

```
C28x Full Context Save/Restore
_____
IntX: ; 8 cycles
       PUSH AR1H:AR0H ; 32-bit
               XAR2 ; 32-bit
XAR3 ; 32-bit
       PUSH
             ; 32-Dit
XAR3 ; 32-bit
XAR4 ; 32-bit
XAR5 ; 32-bit
XAR5 ; 32-bit
XAR6 ; 32-bit
XAR7 ; 32-bit
XT ; 32-bit
T ; 32-bit
       PUSH
       PUSH
        PUSH
        PUSH
        PUSH
        PUSH
        ; + 8 = 16 cycles
        POP
                             XТ
        POP
                             XAR7
        POP
                             XAR6
        POP
                             XAR5
        POP
                             XAR4
        POP
                             XAR3
        POP
                             XAR2
        POP
                             AR1H:AR0H
        IRET
        ; 16 cycles
```

If you perform a task-switch operation (stack changes), the RPC register must be manually saved. You are not to save the RPC register if the stack is not changed.

F.1.3 B0/B1 Memory Map Consideration

Another architecture change to consider is the C27x mapping of blocks B0 and B1. To avoid confusion, on the C28x these blocks are known as M1 and M0 respectively. On the C27x, block B1 was mapped to only data space and block B0 was mapped both in program and data space. In addition, block B0 was mapped to different address ranges in program and in data space. The C27x mapping of these blocks is shown in Figure F–4.

Figure F–4. Mapping of Memory Blocks B0 and B1 on C27x



C27x

On a C28x device at reset, these blocks are mapped uniformly in both program and data space as shown in Figure F–5. This can cause issues when running C27x object code that relies on the C27x mapping. If your code relies on this mapping, you can flip-block M0 and M1 in program space only by clearing the M0M1MAP bit in status register 1 (ST1) to a 0. Executing the "C27MAP" (or "CLRC M0M1MAP") instruction is the only way to clear this bit. With M0M1MAP == 0, the mapping is compatible with the C27x B0 and B1 blocks as shown in Figure D-4. Remember that after a reset M0 and M1 revert to the C28x mapping.

It is strongly recommended that you migrate your code to use the default C28x mapping of these blocks and not rely on the compatible mapping.

Figure F-5. C27x Compatible Mapping of Blocks M0 and M1



Migration From C27x to C28x F-7

F.1.4 C27x Object Compatibility

At reset, the C28x operates in C27x object mode (OBJMODE == 0). In this mode, the C28x CPU is 100% object-code compatible and cycle-count compatible with the C27x. In this case, you will compile your code just as you would for a C27x design as shown in Figure F–6.

Figure F–6. Building a C27x Object File From C27x Source



-v27

Accepts C27x syntax only. Generates C27x object only (assumes OBJMODE = 0)

Once you have taken the mapping of blocks M0 and M1 into account as previously described, you can simply load the C27x object (.out) code into the C28x and run it. When using the C27x compatible mode, you are limited to the C27x instruction set. To take advantage of advanced C28x operations, you should migrate to C28x object code.

When the device is operating in C27x object mode (OBJMODE == 0), the upper bits of the stretched registers (XAR0(31:16) to XAR5(31:16), XAR6(31:22), XAR7(31:22)) are protected from writes. Hence, if the registers are set to zero by a reset then the XARn pointers behave like they do on the C27x and overflow problems are not of concern.

F.2 Moving to a C28x Object

The C28x instruction set is a superset of the C27x instruction set. The syntax of a number of instructions however has changed slightly due to the modifications in registers as previously described. (For a summary of syntax changes, see Section F.3.1 *Instruction Syntax Changes*). To quickly move to C28x object code, the codegen tools allow you to build a C28x object file with a switch allowing for C27x source syntax:

Figure F–7. Building a C28x Object File From Mixed C27x/C28x Source



-v28-m27 Accepts C28x & C27x syntax. Generates C28x object only (assumes OBJMODE == 1)

Prior to running C28x object you must set the mode of the device appropriately (OBJMODE == 1). To do this, you set the OBJMODE bit in ST1 to 1 after reset. This can be done with a "C28OBJ" (or "SETC OBJMODE") instruction. Note that before the "C28OBJ" instruction is executed, the disassembly window in the debugger may display incorrect information. This is because the debugger will decode memory as C27x opcodes until after you execute the "C28OBJ" instruction.

When running in this mode, the disassembly window in your debugger will show the C28x instruction syntax for all instructions. For example, the C27x MOV AR0,@SP instruction will look like MOVZ AR0,@SP, which is the C28x-equivalent instruction.

Now that you are using a C28x object file, you can add C28x operations to your source code.

F.2.1 Caution When Changing OJBMODE

On reset, the XARn registers are forced to 0x0000 0000 and OBJMODE == 0. When operating in C27x compatible mode (OBJMODE == 0), the upper bits of the XARn registers are protected from writes. Some things to be aware of when changing OBJMODE:

- ❑ When operating in C28x object mode (OBJMODE == 1) overflow can occur to the extended portion of XARn registers and program execution is not specified. This would be an issue for assembly code that is reassembled in C28x mode when you relied on the fact that C27x registers were a certain size.
- □ If the user switches to C28x object mode (OBJMODE == 1), then the upper bits of XARn registers may be modified. If you then switch back to C27x

mode (OBJMODE == 0), the upper bits of XARn registers may contain nonzero values. You MUST zero out the upper bits of the XARn registers when switching from OBJMODE == 1 to OBJMODE == 0.

□ It is recommended that you not switch modes frequently in your code. Typically, you will select the appropriate operating mode at boot time and stick to one mode for the whole program.

F.3 Migrating to C28x Object Code

This section describes additional changes to C27x necessary for migrating your C27x code to pure C28x code.

F.3.1 Instruction Syntax Changes

Syntax changes were necessary for clarity and because of changes in the auxiliary registers stretched pointers. Table F–3 shows the C27x instructions that changed syntax on the C28x. For all other C27x instructions, the syntax remains the same. For new C28x instructions, the syntax is documented in Chapter 6.

| C27x Syntax | | | C28x Syntax |
|---------------|---|---------|-------------------------------------|
| ADDB ADDB | ARn,#7bit XAR6/7,#7bit | ADDB | XARn,#7bit |
| SUBB SUBB | ARn,#7bit XAR6/7,#7bit | SUBB | XARn,#7bit |
| MOV | AR0//5,loc16 | MOVZ | AR0//5,loc16 |
| MOVB | AR0//5,#8bit | MOVB | XAR0//5,#8bit |
| MOV MOVL | XAR6/7,loc32 XAR6/7,loc32 | MOVL | XAR6/7,loc32 |
| MOV MOVL | XAR6/7,#22bit XAR6/7,#22bit | MOVL | XAR6/7,#22bit |
| MOV MOVL | loc32,XAR6/7 loc32,XAR6/7 | MOVL | loc32,XAR6/7 |
| CALL LC | 22bit 22bit | LC | 22bit |
| CALL LC | *XAR7 *XAR7 | LC | *XAR7 |
| RET LRET | | LRET | |
| RETE LRETE | | LRETE | |
| MOV | ACC,P {MOVP T,@T decode} | MOVL | ACC,P << PM {MOVP T,@T decode} |
| ADD | ACC,P {MOVA T,@T decode} | ADDL | ACC,P << PM {MOVA T,@T decode} |
| SUB | ACC, P {MOVS T, @T decode} | SUBL | ACC,P << PM {MOVS T,@T decode} |
| CMP | ACC, P | CMPL | ACC,P << PM |
| MOV | P,ACC | MOVL | P,ACC |
| NORM NORM | ACC,ARn++ ACC,XAR6/7++ | NORM | ACC,XARn++ |
| NORM NORM | ACC, ARn ACC, XAR6/7 | NORM | ACC,XARn |
| B SB | 16bitOff {unconditional} 8bitOff {unconditional} | B SB | 16bitOff,UNC [2] 8bitOff,UNC [2] |

Table F-3. Instruction Syntax Change

For conditional branches on the C28x, the UNC code must always be specified for unconditional tests. This will help to distinguish between unconditional C2xLP branches (which have the same mnemonic "B").

F.3.2 Repeatable Instructions

On the C28x, additional instructions have been made repeatable. The following two tables list those instructions that are repeatable on the C28x device. These instructions are repeatable in both C27x compatible mode (OBJMODE = 0) and C28x native mode (OBJMODE = 1). Any instruction that is not listed, which follows a repeat instruction, will execute only once.

C27x operations that were already repeatable include the following:

| ROR | ACC |
|--------|-----------------|
| ROL | ACC |
| NORM | ACC,XARn++ |
| NORM | ACC,XARn |
| SUBCU | ACC,loc16 |
| MAC | P,loc16,0:pma |
| MOV | *(0:addr),loc16 |
| MOV | loc16,*(0:addr) |
| MOV | loc16,#16bit |
| MOV | loc16,#0 |
| PREAD | loc16,*XAR7 |
| PWRITE | *XAR7,loc16 |
| NOP | loc16 |

| MOV | loc16,AX |
|------|-----------------|
| ADD | ACC,loc16 << 16 |
| ADDU | ACC,loc16 |
| SUB | ACC,loc16 << 16 |
| SUBU | ACC,loc16 |
| ADDL | ACC,loc32 |
| SFR | ACC,116 |
| LSL | ACC,116 |
| MOVH | loc16,P |
| MOV | loc16,P |
| MOVA | T,loc16 |
| MOVS | T,loc16 |
| MPYA | P,T,loc16 |
| MPYS | P,T,loc16 |

C27x Operations That Are Made Repeatable On C28x include the following:

F.3.3 Changes to the SUBCU Instruction

The SUBCU instruction changed slightly from the C27x to the C28x. Under the prescribed usage of the SUBCU operation, the change will yield the same result as the C27x.

The SUBCU instruction operates as follows on the C27x device:

To simplify the implementation, the SUBCU operation changed as follows on the C28x:

```
temp(32:0) = ACC << 1 - [loc16] << 16
if( temp(32:0) >= 0 )
    ACC = temp(31:0) + 1;
else
    ACC = ACC << 1;</pre>
```

□ The "temp(32:0)" value is the result of an unsigned 33-bit compare. The carry bit is used to select between \geq or < condition.

- The C flag is affected by the unsigned 33-bit compare operation. The Z, N flags reflect the value in the ACC after the operation is complete. The operation of the C, N, Z flags should be identical to the C27x implementation.
- The V flag and overflow counter (OVC) are not affected by the operation.
 On the C27x the V and OVC flags are affected.

The V and OVC flags may be affected on the C27x and not on the C28x implementation. The values of these flags are not usable under prescribed usage of such an operation.

F.4 Compiling C28x Source Code

Once you move your code to C28x native instructions, you will no longer use the -m27 switch to allow for C27x source as shown in Figure F–8.

Figure F–8. Compiling C28x Source



-v28:

Accepts C28x syntax only. Generates C28x object only (assumes OBJMODE = 1)

Appendix G

Glossary

16-bit operation: An operation that reads or writes 16 bits.32-bit operation: An operation that reads or writes 32 bits.

- **absolute branch:** A branch to an address that is permanently assigned to a memory location. See also *offset branch*.
- ACC: See accumulator (ACC).
- **access:** A term used in this document to mean *read from* or *write to*. For example, to access a register is to read from or write to that register.
- accumulator (ACC): A 32-bit register involved in a majority of the arithmetic and logical calculations done by the C28x. Some instructions that affect ACC use all 32 bits of the register. Others use one of the following portions of ACC: AH (bits 31 through 16), AL (bits 15 through 0), AH.MSB (bits 31 through 24), AH.LSB (bits 23 through 16), AL.MSB (bits 15 through 8), and AL.LSB (bits 7 through 0).
- address-generation logic: Hardware in the CPU that generates the addresses used to fetch instructions or data from memory.
- **address reach:** The range of addresses beginning with 00 0000₁₆ that can be used by a particular addressing mode.
- address register arithmetic unit (ARAU): Hardware in the CPU that generates addresses for values that must be fetched from data memory. The ARAU is also the hardware used to increment or decrement the stack pointer (SP) and the auxiliary registers (AR0, AR1, AR2, AR3, AR4, AR5, XAR6, and XAR7).
- **addressing mode:** The method by which an instruction interprets its operands to acquire the data and/or addresses it needs.
- **AH:** *High word of the accumulator.* The name given to bits 31 through 16 of the accumulator.

- AH.LSB: Least significant byte of AH. The name given to bits 23 through 16 of the accumulator.
- **AH.MSB:** Most significant byte of AH. The name given to bits 31 through 24 of the accumulator.
- AL: Low word of the accumulator. The name given to bits 15 through 0 of the accumulator.
- AL.LSB: Least significant byte of AL. The name given to bits 7 through 0 of the accumulator.
- **AL.MSB:** *Most significant byte of AL*. The name given to bits 15 through 8 of the accumulator.
- ALU: See arithmetic logic unit (ALU).
- **analysis logic:** A portion of the emulation logic in the core. The analysis logic is responsible for managing the following debug activities: hardware breakpoints, hardware watchpoints, data logging, and benchmark/event counting.
- **approve an interrupt request:** Allow an interrupt to be serviced. If the interrupt is maskable, the CPU approves the request only if it is properly enabled. If the interrupt is nonmaskable, the CPU approves the request immediately. See also *interrupt request* and *service an interrupt*.
- **ARAU:** See address register arithmetic unit (ARAU).
- **arithmetic logic unit (ALU):** A 32-bit hardware unit in the CPU that performs 2s-complement arithmetic and Boolean logic operations. The ALU accepts inputs from data from registers, from data memory, or from the program control logic. The ALU sends results to a register or to data memory.
- arithmetic shift: A shift that treats the shifted value as signed. See also *logical shift*.
- **ARP:** See auxiliary register pointer (ARP).
- **ARP indirect addressing mode:** The indirect addressing mode that uses the current auxiliary register to point to a location in data space. The current auxiliary register is the auxiliary register pointed to by the ARP. See also *auxiliary register pointer (ARP)*.
- **automatic context save:** A save of system context (modes and key register values) performed by the CPU just prior to executing an interrupt service routine. See also *context save*.

- **auxiliary register:** One of eight registers used as a pointer to a memory location. The register is operated on by the auxiliary register arithmetic unit (ARAU) and is selected by the auxiliary register pointer (ARP). See also *AR0–AR5*, *AR6/AR7*, and *XAR6/XAR7*.
- **auxiliary-register indirect addressing mode:** The indirect addressing mode that allows you to use the name of an auxiliary register in an operand that uses that register as a pointer. See also *ARP indirect addressing mode*.
- **auxiliary register pointer (ARP):** A 3-bit field in status register ST1 that selects the current auxiliary register. When an instruction uses ARP indirect addressing mode, that instruction uses the current auxiliary register to point to data space. When an instruction specifies auxiliary register *n* by using auxiliary-register indirect addressing mode, the ARP is updated, so that it points to auxiliary register *n*. See also *current auxiliary register*.

- **background code:** The body of code that can be halted during debugging because it is not time-critical.
- **barrel shifter:** Hardware in the CPU that performs all left and right shifts of register or data-space values.
- **bit field:** One or more register bits that are differentiated from other bits in the same register by a specific name and function.
- **bit manipulation:** The testing or modifying of individual bits in a register or data-space location.
- **boundary scan:** The use of scan registers on the border of a chip or section of logic to capture the pin states. By scanning these registers, all pin states can be transmitted through the JTAG port for analysis.
- **branch:** 1) A forcing of program control to a new address. 2) An instruction that forces program control to a new address but neither saves a return address (like a call) nor restores a return address (like a return).
- **break event:** A debug event that causes the CPU to enter the debug-halt state.
- **breakpoint:** A place in a routine specified by a breakpoint instruction or hardware breakpoint, where the execution of the routine is to be halted and the debug-halt state entered.

С

C bit: See carry (C) bit.

- **call:** 1) The operation of saving a return address and then forcing program control to a new address. 2) An instruction that performs such an operation. See also *return*.
- **carry (C) bit:** A bit in status register ST0 that reflects whether an addition has generated a carry or a subtraction has generated a borrow.
- **circular addressing mode:** The indirect addressing mode that can be used to implement a circular buffer.
- **circular buffer:** A block of addresses referenced by a pointer using circular addressing mode, so that each time the pointer reaches the bottom of the block, the pointer is modified to point back to the top of the block.
- **clear :** To clear a bit is to write a 0 to it. To clear a register or memory location is to load all its bits with 0s. See also *set*.
- **COFF:** Common object file format. A binary object file format that promotes modular programming by supporting the concept of sections, where a section is a relocatable block of code or data that ultimately occupies a space adjacent to other blocks of code in the memory map.
- **conditional branch instruction:** A branch instruction that may or may not cause a branch, depending on a specified or predefined condition (for example, the state of a bit).
- **context restore:** A restoring of the previous state of a system (for example, modes and key register values) prior to returning from a subroutine. See also *context save*.
- **context save:** A save of the current state of a system (for example, modes and key register values) prior to executing the main body of a subroutine that requires a different context. See also *context restore*.
- **core:** The portion of the C28x that consists of a CPU, a block of emulation circuitry, and a set of signals for interfacing with memory and peripheral devices.
- **current auxiliary register:** The register selected by the auxiliary register pointer (ARP) in status register. For example, if ARP = 3, the current auxiliary register is AR3. See also *auxiliary registers*.
- **current data page:** The data page selected by the data page pointer. For example, if DP = 0, the current data page is 0. See also *data page*.

D

- D1 phase: See decode 1 (D1) phase.
- D2 phase: See decode 2 (D2) phase.
- **data logging:** Transferring one or more packets of data from CPU registers or memory to an external host processor.
- data log interrupt (DLOGINT): A maskable interrupt triggered by the onchip emulation logic when a data logging transfer has been completed.
- **data page:** A 64-word portion of the total 4M words of data space. Each data page has a specific start address and end address. See also *data page pointer (DP)* and *current data page.*
- **data page pointer (DP):** A 16-bit pointer that identifies which 64-word data page is accessed in DP direct addressing mode. For example, for as long as DP = 500, instructions that use DP direct addressing mode will access data page 500.
- data-/program-write data bus (DWDB): The bus that carries data during writes to data space or program space.
- data-read address bus (DRAB): The bus that carries addresses for reads from data space.
- data-read data bus (DRDB): The bus that carries data during reads from data space.
- data-write address bus (DWAB): The bus that carries addresses for writes to data space.
- **DBGIER:** See debug interrupt enable register (DBGIER).
- DBGM bit: See debug enable mask (DBGM) bit.
- **DBGSTAT:** See debug status register (DBGSTAT).
- **debug-and-test direct memory access (DT–DMA):** An access of a register or memory location to provide visibility to this location during debugging. The access is performed with variable levels of intrusiveness by a hardware DT-DMA mechanism inside the core.
- **debug enable mask (DBGM) bit:** A bit in status register ST1 used to enable (DBGM = 0) or disable (DBGM = 1) debug events such as analysis breakpoints or debug-and-test direct memory accesses (DT-DMAs).

- **debug event:** An action such as the decoding of a software breakpoint instruction, the occurrence of an analysis breakpoint/watchpoint, or a request from a host processor that may result in special debug behavior, such as halting the device or pulsing one of the debug interface signals EMU0 or EMU1. See also *break event* and *debug enable mask (DBGM) bit*.
- **debug-halt state:** A debug execution state that is entered through a break event. In this state the CPU is halted. See also *single-instruction state* and *run state*.
- debug host: See host processor.
- **debug interrupt enable register (DBGIER):** The register that determines which of the maskable interrupts are time-critical when the CPU is halted in real-time mode. If a bit in the DBGIER is 1, the corresponding interrupt is time-critical/enabled; otherwise, it is disabled. Time-critical interrupts also must be enabled in the interrupt enable register (IER) to be serviced.
- **debug status register (DBGSTAT):** A register that holds special debug status information. This register, which need not be read from or written to, is saved and restored during interrupt servicing, to preserve the debug context during debugging.
- **decode an instruction:** To identify an instruction and prepare the CPU to perform the operation the instruction requires.
- **decode 1 (D1) phase:** The third of eight pipeline phases an instruction passes through. In this phase, the CPU identifies instruction boundaries in the instruction-fetch queue and determines whether the next instruction to be executed is an illegal instruction. See also *pipeline phases*.
- **decode 2 (D2) phase:** The fourth of eight pipeline phases an instruction passes through. In this phase, the CPU accepts an instruction from the instruction-fetch queue and completes the decoding of that instruction, performing such activities as address generation and pointer modification. See also *pipeline phases*.
- **decrement:** To subtract 1 or 2 from a register or memory value. The value subtracted depends on the circumstance. For example, if you use the operand *--AR4, the auxiliary register AR4 is decremented by 1 for a 16-bit operation and by 2 for a 32-bit operation.

device reset: See reset.

- **direct addressing modes:** The addressing modes that access data space as if it were 65 536 separate blocks of 64 words each. DP direct addressing mode uses the data page pointer (DP) to select a data page from 0 to 65 535. PAGE0 direct addressing mode uses data page 0, regardless of the value in the DP.
- **discontinuity:** See program-flow discontinuity.
- **DLOGINT:** See data log interrupt (DLOGINT).
- **DP:** See data page pointer (DP).
- **DP direct addressing mode:** A direct addressing mode that uses the data page pointer (DP) to select a data page from 0 to 65 535. See also *PAGE0 direct addressing mode*.
- **DRAB:** See data-read address bus (DRAB).
- **DRDB:** See data-read data bus (DRDB).
- **DT–DMA:** See debug-and-test direct memory access (DT-DMA).
- **DWAB:** See data-write address bus (DWAB).
- **DWDB:** See data-/program-write data bus (DWDB).
- **E phase:** See *execute* (E) *phase*.

- EALLOW bit: See emulation access enable (EALLOW) bit.
- **EMU0 and EMU1 pins:** Pins known as the TI extensions to the JTAG interface. These pins can be used as either inputs or outputs and are available to help monitor and control an emulation target system that is using a JTAG interface.
- emulation access enable (EALLOW) bit: A bit in status register ST1 that enables (EALLOW = 1) or disables (EALLOW = 0) access to the emulation registers. The EALLOW instruction sets the EALLOW bit, and the EDIS instruction clears the EALLOW bit.
- **emulation logic:** The block of hardware in the core that is responsible controlling emulation activities such as data logging and switching among debug execution states.
- **emulation registers:** Memory-mapped registers that are available for controlling and monitoring emulation activities.

enable bit: See interrupt enable bits.

- **execute an instruction:** Take an instruction from the decode 2 phase of the pipeline through the write phase of the pipeline.
- **execute (E) phase:** The seventh of eight pipeline phases an instruction passes through. In this phase, the CPU performs all multiplier, shifter, and arithmetic-logic-unit (ALU) operations. See also *pipeline phases*.

extended auxiliary registers: See XAR6/XAR7.

F

F1 phase: See fetch 1 (F1) phase.

- F2 phase: See fetch 2 (F2) phase.
- FC: See fetch counter (FC).
- fetch 1 (F1) phase: The first of eight pipeline phases an instruction passes through. In this phase, the CPU places on the program-read bus the address of the instruction(s) to be fetched. See also *pipeline phases*.
- fetch 2 (F2) phase: The second of eight pipeline phases an instruction passes through. In this phase, the CPU fetches an instruction or instructions from program memory. See also *pipeline phases*.
- **fetch counter (FC) :** The register that contains the address of the instruction that is being fetched from program memory.
- field : See bit field.

- **hardware interrupt:** An interrupt initiated by a physical signal (for example, from a pin or from the emulation logic). See also *software interrupt*.
- **hardware interrupt priority:** A priority ranking used by the CPU to determine the order in which simultaneously occurring hardware interrupts are serviced.
- hardware reset: See reset.
- **high addresses:** Addresses closer to 3F FFFF₁₆ than to 00 0000₁₆. See also *low addresses*.
- high bits: See MSB.

high word: The 16 MSBs of a 32-bit value. See also low word.

host processor: The processor running the user interface for a debugger.

- IC: See instruction counter (IC).
- **IDLESTAT (IDLE status) bit:** A bit in status register ST1 that indicates when an IDLE instruction has the CPU in the idle state (IDLESTAT = 1).
- idle state: The low-power state the CPU enters when it executes the IDLE instruction.
- **IEEE 1149.1 standard:** "IEEE Standard Test Access Port and Boundary-Scan Architecture", first released in 1990. See also *JTAG*.
- **IER:** See interrupt enable register (IER).
- **IFR:** See interrupt flag register (IFR).
- **illegal instruction:** An unacceptable value read from program memory during an instruction fetch. Unacceptable values are 0000₁₆, FFFF₁₆, or any value that does not match a defined opcode.
- **illegal-instruction trap:** A trap that is serviced when an illegal instruction is decoded.
- **immediate address:** An address that is specified directly in an instruction as a constant.
- **immediate addressing modes:** Addressing modes that accept a constant as an operand.
- **immediate constant/data:** A constant specified directly as an operand of an instruction.
- **immediate-constant addressing mode:** An immediate addressing mode that accepts a constant as an operand and interprets that constant as data to be stored or processed.
- **immediate-pointer addressing mode:** An immediate addressing mode that accepts a constant as an operand and interprets that constant as the 16 LSBs of a 22-bit address. The six MSBs of the address are filled with 0s.
- **increment:** To add 1 or 2 to a register or memory value. The value added depends on the circumstance. For example, if you use the operand *AR4++, the auxiliary register AR4 is incremented by 1 for a 16-bit operation and by 2 for a 32-bit operation.

- indirect addressing modes: Addressing modes that use pointers to access memory. The available pointers are auxiliary registers AR0–AR5, extended auxiliary registers XAR6 and XAR7, and the stack pointer (SP).
- **instruction boundary:** The point where the CPU has finished one instruction and is considering what it will do next — move on to the next instruction.
- instruction counter (IC): The register that points to the instruction in the decode 1 phase (the instruction that is to enter the decode 2 phase next). Also, on an interrupt or call operation, the IC value represents the return address, which is saved to the stack or to auxiliary register XAR7.
- **instruction-fetch mechanism:** The hardware for the fetch 1 and fetch 2 phases of the pipeline. This hardware is responsible for fetching instructions from program memory and filling an instruction-fetch queue.
- **instruction-fetch queue:** A queue of four 32-bit registers that receives fetched instructions and holds them for decoding. When a program-flow discontinuity occurs, the instruction-fetch queue is emptied.
- **instruction-not-available condition:** The condition that occurs when the decode 2 pipeline hardware requests an instruction but there are no instructions waiting in the instruction-fetch queue. This condition causes the decode 2 through write phases of the pipeline to freeze until one or more new instructions have been fetched.
- **instruction register:** The register that contains the instruction that has reached the decode 2 pipeline phase.
- **instruction word:** Either an entire 16-bit opcode or one of the halves of a 32-bit opcode.
- **INT1–INT14:** Fourteen general-purpose interrupts that are triggered by signals at pins of the same names. These interrupts are maskable and have corresponding bits in the interrupt flag register (IFR), the interrupt enable register (IER), and the debug interrupt enable register (DBGIER).
- **Interrupt boundary:** An instruction boundary where the CPU can insert an interrupt between two instructions. See also *instruction boundary*.
- **interrupt enable bits:** Bits responsible for enabling or disabling maskable interrupts. The enable bits are all the bits in the interrupt enable register (IER), all the bits in the debug interrupt enable register (DBGIER), and the interrupt global mask bit (INTM in status register ST1).
- **interrupt enable register (IER):** Each of the maskable interrupts has an interrupt enable bit in this register. If a bit in the IER is 1, the corresponding interrupt is enabled; otherwise, it is disabled. See also *debug interrupt enable register (DBGIER)*.

- **interrupt flag bit:** A bit in the interrupt flag register (IFR). If the interrupt flag bit is 1, the corresponding interrupt has been requested by hardware and is awaiting approval by the CPU.
- **interrupt flag register (IFR):** The register that contains the interrupt flag bits for the maskable interrupts. If a bit in the IFR is 1, the corresponding interrupt has been requested by hardware and is awaiting approval by the CPU.
- interrupt global mask (INTM) bit: A bit in status register ST1 that globally enables or disables the maskable interrupts. If an interrupt is enabled in the interrupt enable register (IER) but not by the INTM bit, it is not serviced. The only time this bit is ignored is when the CPU is in real-time mode and is in the debug-halt state; in this situation, the interrupt must be enabled in the IER and in the DBGIER (debug interrupt enable register).
- interrupt priority: See hardware interrupt priority.
- **interrupt request:** A signal or instruction that requests the CPU to execute a particular interrupt service routine. See also *approve an interrupt request* and *service an interrupt*.
- **interrupt service routine (ISR):** A subroutine that is linked to a specific interrupt by way of an interrupt vector.
- **interrupt vector:** The start address of an interrupt service routine. After approving an interrupt request, the CPU fetches the interrupt vector from your interrupt vector table and uses the vector to branch to the start of the corresponding interrupt service routine.
- **interrupt vector location:** The preset location in program memory where an interrupt vector must reside.
- **interrupt vector table:** The list of interrupt vectors you assign in program memory.
- **INTM bit:** See interrupt global mask (INTM) bit.
- **ISR:** See interrupt service routine (ISR).

J

JTAG: Joint Test Action Group. The Joint Test Action Group was formed in 1985 to develop economical test methodologies for systems designed around complex integrated circuits and assembled with surface-mount technologies. The group drafted a standard that was subsequently adopted by IEEE as IEEE Standard 1149.1-1990, "IEEE Standard Test Access Port and Boundary-Scan Architecture". See also *boundary scan*; *test access port (TAP)*.

JTAG port: See test access port (TAP).

- **latch:** Hold a bit at the same value until a given event occurs. For example, when an overflow occurs in the accumulator, the V bit is set and latched at 1 until it is cleared by a conditional branch instruction or by a write to status register ST0. An interrupt is latched when its flag bit has been latched in the interrupt flag register (IFR).
- **least significant bit (LSB):** The bit in the lowest position of a binary number. For example, the LSB of a 16-bit register value is bit 0. See also *MSB*, *LSByte*, and *MSByte*.
- **least significant byte (LSByte):** The byte in the lowest position of a binary value. The LSByte of a value consists of the eight LSBs. See also *MSByte*, *LSB*, and *MSB*.
- **location:** A space where data can reside. A location may be a CPU register or a space in memory.
- **logical shift:** A shift that treats the shifted value as unsigned. See also *arithmetic shift*.
- **LOOP (loop instruction status) bit:** A bit in status register ST1 that indicates when a LOOPNZ or LOOPZ instruction is being executed (LOOP = 1).
- **low addresses:** Addresses closer to 00 0000₁₆ than to 3F FFFF₁₆. See also *high addresses*.

low bits: See LSB.

low word: The 16 LSBs of a 32-bit value. See also high word.

- **LSB:** When used in a syntax of the MOVB instruction, LSB means least significant byte. Otherwise, LSB means least significant bit. See *least significant bit (LSB)* and *least significant byte (LSByte)*.
- **LSByte:** See *least significant byte (LSByte)*.

Μ

- **maskable interrupt:** An interrupt that can be disabled by software so that the CPU does not service it until it is enabled by software. See also *non-maskable interrupt*.
- **memory interface:** The buses and signals responsible for carrying communications between the core and on-chip memory/peripherals.
- **memory-mapped register:** A register that can be accessed at addresses in data space.
- **memory wrapper:** The hardware around a memory block that identifies access requests and controls accesses for that memory block.
- **mirror:** A range of addresses that is the same size and is mapped to the same physical memory block as another range of addresses.
- **most significant bit (MSB):** The bit in the highest position of a binary number. For example, the MSB of a 16-bit register value is bit 15. See also *LSB*, *LSByte*, and *MSByte*.
- **most significant byte (MSByte):** The byte in the highest position of a binary value. The MSByte of a value consists of the eight MSBs. See also *LSByte, LSB*, and *MSB*.
- **MSB:** When used in a syntax of the MOVB instruction, MSB means most significant byte. Otherwise MSB means most significant bit. See *most significant bit (MSB)* and *most significant byte (MSByte)*.
- **MSByte:** See most significant byte (MSByte).
- **multiplicand register (T):** The primary function of this register, also called the T register, is to hold one of the values to be multiplied during a multiplication. The following shift instructions use the four LSBs to hold the shift count: ASR (arithmetic shift right), LSL (logical shift left), LSR (logical shift right), and SFR (shift accumulator right). The T register can also be used as a general-purpose 16-bit register.

Ν

- N (negative flag) bit: A bit in status register ST0 that indicates whether the result of a calculation is a negative number (N = 1). N is set to match the MSB of the result.
- nested interrupt: An interrupt that occurs within an interrupt service routine.
- **NMI:** A hardware interrupt that is nonmaskable, like reset (RS), but does not reset the CPU. <u>NMI</u> simply forces the CPU to execute its interrupt service routine.
- **nonmaskable interrupt:** An interrupt that cannot be blocked by software and is approved by the CPU immediately. See also *maskable interrupt*.

offset branch: A branch that uses a specified or generated offset value to jump to an address relative to the current position of the program counter (PC). See also *absolute branch*.

- **opcode:** This document uses opcode to mean the complete code for an instruction. Thus, an opcode includes the binary sequence for the instruction type and the binary sequence and/or constant in which the operands are encoded.
- **operand :** This document uses operand to mean one of the values entered after the instruction mnemonic and separated by commas (or for a shift operand, separated by the symbol <<). For example, in the CLRC INTM instruction, CLRC is the mnemonic and INTM is the operand.
- operation: 1) A defined action; namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result of any permitted combination of operands. 2) The set of such acts specified by a rule, or the rule itself. 3) The act specified by a single computer instruction. 4) A program step undertaken or executed by a computer; for example, addition, multiplication, extraction, comparison, shift, transfer, etc. 5) The specific action performed by a logic element.
- **OVC:** See overflow counter (OVC).
- **OVM:** See overflow mode (OVM) bit.

0

- **overflow counter (OVC):** A 6-bit counter in status register ST0 that can be used to track overflows in the accumulator (ACC). The OVC is enabled only when the overflow mode (OVM) bit in ST0 is 0. When OVM = 0, the OVC is incremented by 1 for every overflow in the positive direction (too large a positive number) and decremented by 1 for every overflow in the negative direction (too large a negative number). The saturate (SAT) instruction modifies ACC to reflect the net overflow represented in the OVC.
- **overflow flag (V):** A bit in status register ST0 that indicates when the result of an operation causes an overflow in the location holding the result (V = 1). If no overflow occurs, V is not modified.
- **overflow mode (OVM) bit:** A bit in the status register ST0 that enables or disables overflow mode. When overflow mode is on (OVM = 1) and an overflow occurs, the CPU fills the accumulator (ACC) with a saturation value. When overflow mode is off (OVM = 0), the CPU lets ACC overflow normally but keeps track of each overflow by incrementing or decrementing by 1 the overflow counter (OVC) in ST0.

P register: See product register (P).

- **PAB:** See program address bus (PAB).
- **PAGE0 bit:** *PAGE0 addressing mode configuration bit.* This bit, in status register ST1, selects between two addressing modes: PAGE0 stack addressing mode (PAGE = 0) and PAGE0 direct addressing mode (PAGE0 = 1).
- **PAGE0 direct addressing mode:** The direct addressing mode that uses data page 0 regardless of the value in the data page pointer (DP). This mode is available only when the PAGE0 bit in status register ST1 is 1. See also *DP direct addressing mode* and *PAGE0 stack addressing mode*.
- **PAGE0 stack addressing mode:** The indirect addressing mode that references a value on the stack by subtracting a 6-bit offset from the current position of the stack pointer (SP). This mode is available only when the PAGE0 bit in status register ST1 is 0. See also *stack-pointer indirect addressing mode*.
- **PC:** See program counter (PC).
- **pending interrupt:** An interrupt that has been requested but is waiting for approval from the CPU. See also *approve an interrupt request.*

- **peripheral-interface logic:** Hardware that is responsible for handling communications between a processor and a peripheral.
- PH: The high word (16 MSBs) of the P register.
- phases: See pipeline phases.
- **pipeline:** The hardware in the CPU that takes each instruction through eight independent phases for fetching, decoding, and executing. During any given CPU cycle, there can be up to eight instructions in the pipeline, each at a different phase of completion. The phases, listed in the order in which instructions pass through them, are fetch 1, fetch 2, decode 1, decode 2, read 1, read 2, execute, and write.
- **pipeline conflict:** A situation in which two instructions in the pipeline try to access a register or memory location out of order, causing improper code operation. The C28x pipeline inserts as many inactive cycles as needed between conflicting instructions to prevent pipeline conflicts.
- **pipeline freeze:** A halt in pipeline activity in one of the two decoupled portions of the pipeline. Freezes in the fetch 1 through decode 1 portion of the pipeline are caused by a not-ready signal from program memory. Freezes in the decode 2 through write portion are caused by lack of instructions in the instruction-fetch queue or by not-ready signals from memory.
- **pipeline phases:** The eight stages an instruction must pass through to be fetched, decoded, and executed. The phases, listed in the order in which instructions pass through them, are fetch 1, fetch 2, decode 1, decode 2, read 1, read 2, execute, and write.
- **pipeline-protection mechanism:** The mechanism responsible for identifying potential pipeline conflicts and preventing them by adding inactive cycles between the conflicting instructions.
- PL: The low word (16 LSBs) of the P register.
- **PM bits:** See product shift mode (PM) bits.
- **PRDB:** See program-read data bus (PRDB).
- priority: See interrupt priority.
- **product register (P):** This register, also called the P register, is given the results of most multiplications done by the CPU. The only other register that can be given the result of a multiplication is the accumulator (ACC). See also *PH* and *PL*.

- **product shift mode (PM) bits:** A 3-bit field in status register ST0 that enables you to select one of eight product shift modes. The product shift mode determines whether or how the P register value is shifted before being used by an instruction. You have the choices of a left shift by 1 bit, no shift, or a right shift by N, where N is a number from 1 to 6.
- **program address bus (PAB):** The bus that carries addresses for reads and writes from program space.
- **program address generation logic:** This logic generates the addresses used to fetch instructions or data from program memory and places each address on the program address bus (PAB).
- **program control logic:** This logic stores a queue of instructions that have been fetched from program memory by way of the program-read bus (PRDB). It also decodes these instructions and passes commands and constant data to other parts of the CPU.
- **program counter (PC):** When the pipeline is full, the 22-bit PC always points to the instruction that is currently being processed—the instruction that has just reached the decode 2 phase of the pipeline.
- **program-flow discontinuity:** A branching to a nonsequential address caused by a branch, a call, an interrupt, a return, or the repetition of an instruction.
- **program-read data bus (PRDB):** The bus that carries instructions or data during reads from program space.

R

- **R1 phase:** See read 1 (R1) phase.
- R2 phase: See read 2 (R2) phase.
- **read 1 (R1) phase:** The fifth of eight pipeline phases an instruction passes through. In this phase, if data is to be read from memory, the CPU drives the address(es) on the appropriate address bus(es). See also *pipeline phases*.
- **read 2 (R2) phase:** The sixth of eight pipeline phases an instruction passes through. In this phase, data addressed in the read 1 phase is fetched from memory. See also *pipeline phases*.

- **ready signals:** When the core requests a read from or write to a memory device or peripheral device, that device can take more time to finish the data transfer than the core allots by default. Each device must use one of the core's *ready signals* to insert wait states into the data transfer when it needs more time. Wait-state requests freeze a portion of the pipeline if they are received during the fetch 1, read 1, or write pipeline phase of an instruction.
- **real-time mode:** An emulation mode that enables you execute certain interrupts (time-critical interrupts), even when the CPU is halted. See also *stop mode*.
- real-time operating system interrupt (RTOSINT): A maskable hardware interrupt generated by the emulation hardware in response to certain debug events. This interrupt should be disabled in the interrupt enable register (IER) and the debug interrupt enable register (DBGIER) unless there is a real-time operating system present in your debug system.
- **reduced instruction set computer (RISC):** A computer whose instruction set and related decode mechanism are much simpler than those of microprogrammed complex instruction set computers.
- **register addressing mode:** An addressing mode that enables you to reference registers by name.
- **register conflict:** A pipeline conflict that would occur if an instruction read a register value before that value were changed by a prior instruction. The C28x pipeline inserts as many inactive cycles as needed between conflicting instructions to prevent register conflicts.
- **register pair:** One of the pairs of CPU register stored to the stack during an automatic context save.
- **repeat counter (RPTC):** The counter that is loaded by the RPT (repeat) instruction. The number in the counter is the number of times the instruction qualified by RPT is to be repeated after its initial execution.
- reserved: A term used to describe memory locations or other items that you cannot use or modify.
- **reset:** To return the DSP to a known state; an action initiated by the reset (RS) signal.
- **return:** 1) The operation of forcing program control to a return address. 2) An instruction that performs such an operation. See also *call*.
- **return address:** The address at which the CPU resumes processing after executing a subroutine or interrupt service routine.

- **RISC:** See reduced instruction set computer (RISC).
- **rotate operation:** An operation performed by the ROL (rotate accumulator left) or ROR (rotate accumulator right) instruction. The operation, which involves a shift by 1 bit, can be seen as the rotation of a 33-bit value that is the concatenation of the carry bit (C) and the accumulator (ACC).
- **RPTC:** See *repeat counter (RPTC)*.
- **RTOSINT :** See real-time operating system interrupt (RTOSINT).
- **RUN command :** A debugger command used to execute all or a portion of a program. The RUN 1 command causes the debugger to execute a single instruction.
- **run state:** A debug execution state. In this state, the CPU is executing code and servicing interrupts freely. See also *debug-halt state* and *single-in-struction state*.

- **select signal:** An output signal from the C28x that can be used to select specific memory or peripheral devices for particular types of read and write operations.
- **scan controller:** A device that performs JTAG state sequences sent to it by a host processor. These sequences, in turn, control the operation of a target device.
- **service an interrupt :** The CPU services an interrupt by preparing for and then executing the corresponding interrupt service routine. See also *interrupt request* and *approve an interrupt request*.
- set: To set a bit is to write a 1 to it. If a bit is set, it contains 1. See also *clear*.
- **sign extend:** To fill the unused most significant bits (MSBs) of a value with copies of the value's sign bit.
- sign-extension mode (SXM) bit: A bit in status register ST0 that enables or suppresses sign extension. When sign-extension is enabled (SXM = 1), operands of certain instructions are treated as signed and are sign extended during shifting.
- **single-instruction state:** A debug execution state. In this state, the CPU executes one instruction and then returns to the debug-halt state. See also *debug-halt state* and *run state*.

16-bit operation: An operation that reads or writes 16 bits.

- **software interrupt:** An interrupt initiated by an instruction. See also *hard-ware interrupt*.
- SP: See stack pointer (SP).
- SPA bit: See stack pointer alignment (SPA) bit.
- **ST0:** See status registers ST0 and ST1.
- ST1: See status registers ST0 and ST1.
- stack : The C28x stack is a software stack implemented by the use of a stack pointer (SP). The SP, a 16-bit CPU register, can be used to reference a value in the first 64K words of data memory (addresses 00 0000₁₆-00 FFFF₁₆).
- stack pointer (SP): A 16-bit CPU register that enables you to use any portion of the first 64K words of data memory as a software stack. The SP always points to the next empty location in the stack.
- **stack pointer alignment (SPA) bit:** A bit in status register ST1 that indicates whether an ASP instruction has forced the SP to align to the next even address (SPA = 1).
- **stack-pointer indirect addressing mode:** The indirect addressing mode that references a data-memory value at the current position of the stack pointer (SP). See also *PAGE0 stack addressing mode*.
- status registers ST0 and ST1: These CPU registers contain control bits that affect the operation of the C28x and contain flag bits that reflect the results of operations.
- **STEP command:** A debugger command that causes the debugger to single-step through a program. The STEP1 command causes the debugger to execute a single instruction.
- **stop mode:** An emulation mode that provides complete control of program execution. When the CPU is halted in stop mode, all interrupts (including reset and nonmaskable interrupts) are ignored until the CPU receives a directive to run code again. See also *real-time mode*.
- **suppress sign extension:** Prevent sign extension from occurring during a shift operation. See also *sign extend*.
- **SXM bit:** See sign-extension mode (SXM) bit.

Т

- **T register:** The primary function of this register, also called the multiplicand register, is to hold one of the values to be multiplied during a multiplication. The following shift instructions use the four LSBs to hold the shift count: ASR (arithmetic shift right), LSL (logical shift left), LSR (logical shift right), and SFR (shift accumulator right). The T register can also be used as a general-purpose 16-bit register.
- **TAP:** See test access port (TAP).
- target device/system: The device/system on which the code you have developed is executed.
- **TC bit:** See *test/control flag (TC)*.
- **test access port (TAP):** A standard communication port defined by IEEE standard 1149.1–1990 included in the DSP to implement boundary scan functions and/or to provide communication between the DSP and emulator.
- **test/control flag (TC):** A bit in status register ST0 that shows the result of a test performed by the TBIT (test bit) instruction or the NORM (normalize) instruction.
- **test-logic-reset:** A test and emulation logic condition that occurs when the TRST signal is pulled low or when the TMS signal is used to advance the JTAG state machine to the TLR state. This logic is a different type than that used by the CPU, which resets functional logic.
- 32-bit operation: An operation that reads or writes 32 bits.
- TI extension pins: See EMU0 and EMU1 pins.
- **time-critical interrupt:** An interrupt that must be serviced even when background code is halted. For example, a time-critical interrupt might service a motor controller or a high-speed timer. See also *debug interrupt enable register (DBGIER)*.

U

USER1–USER12 interrupts: The interrupt vector table contains twelve locations for user-defined software interrupts. These interrupts, called USER1–USER12 in this document, can be initiated only by way of the TRAP instruction.

V

- V bit (overflow flag): A bit in status register ST0 that indicates when the result of an operation causes an overflow in the location holding the result (V = 1). If no overflow occurs, V is not modified.
- vector: See interrupt vector.
- vector location: See interrupt vector location.
- **vector map (VMAP) bit:** A bit in status register ST1 that determines the addresses to which the interrupt vectors are mapped. When VMAP = 0, the interrupt vectors are mapped to addresses $00\ 0000_{16}-00\ 003F_{16}$ in program memory. When VMAP = 1, the vectors are mapped to addresses 3F FFC0_{16}-3F FFFF_{16} in program memory.
- vector table: See interrupt vector table.

W phase: See write (W) phase.

- **wait state:** A cycle during which the CPU waits for a memory or peripheral device to be ready for a read or write operation.
- **watchpoint:** A place in a routine where it is to be halted if an address or an address and data combination match specified compare values. When a watchpoint is reached, the routine is halted and the CPU enters the debug-halt state.
- **word:** In this document, a word is 16 bits unless specifically stated to be otherwise.
- write (W) phase: The last of eight pipeline phases an instruction passes through. In this phase, if a value or result is to be written to memory, the CPU sends to memory the destination address and the data to be written. See also *pipeline phases*.

Ζ

- **zero fill:** Fill the unused low- and/or high-order bits of a value with 0s.
- **zero flag (Z):** A bit in status register ST0 that indicates when the result of an operation is 0 (Z = 1).

W

Index

Α

ABORTI 6-18 ABORTI instruction 7-15 ABS ACC 6-19 ABSTC ACC 6-20 access to CPU registers during emulation 7-16 access to memory during emulation 7-16 accesses polite 7-16 rude 7-16 Accumulator C-4 accumulator 2-6 AH (high word) 2-6 AH.LSB 2-7 AH.MSB 2-7 AL (low word) 2-6 AL.LSB 2-7 AL.MSB 2-7 portions that are individually accessible 2-7 ADD ACC,#16bit<#0..15 6-22 ADD ACC,loc16< T 6-24 ADD ACC,loc16<#0 6-25 ADD AX, loc16 6-27 ADD loc16, AX 6-28 ADD loc16,#16bitSigned 6-29 ADDB ACC,#8bit 6-30 ADDB AX, #8bitSigned 6-31 ADDB SP, #7bit 6-32 ADDB XARn, #7bit 6-33 ADDCL ACC,loc32 6-34 ADDCU ACC,loc16 6-35 ADDL ACC,loc32 6-36 ADDL ACC,P < PM 6-37 ADDL loc32, ACC 6-38

address buses 1-9 address counters FC, IC, and PC 4-5 address maps 1-8 address reach C-5 address register arithmetic unit (ARAU) 1-5, 2-2 addressing modes 5-1, 5-2 byte 5-31 direct 2-10 direct addressing 5-2 indirect 2-12 indirect addressing 5-2 program space 5-30 register 5-25 stack addressing 5-2 Addressing Modes for "loc16" or "loc32", table 5-4 Addressing Modes Select Bit (AMODE) 5-4 ADDRH register 7-24 ADDRL register 7-24 ADDU ACC,loc16 6-39 ADDUL P,loc32 6-40 ADDUL ACC, loc32 6-41 ADRK #8bit 6-42 AH (high word of accumulator) 2-6 AL (low word of accumulator) 2-6 align stack pointer 6-52 AL.LSB (part of accumulator) 2-7 AL.MSB (part of accumulator) 2-7 AMODE 5-4, C-9, F-4 AMODE bit 1-2 analysis resources breakpoints 7-19 clearing resources 7-30 counters 7-20 data logging 7-23 sharing resources 7-30 watchpoints 7-19 AND ACC, #16bit < #0..15 6-43

AND ACC, loc16 6-43, 6-44 AND AX, loc16, #16bit 6-45 AND IER,#16bit 6-46 AND IFR.#16bit 6-47 AND loc16, AX 6-48 AND AX, loc16 6-49 AND IER and OR IER instructions, note about RTOSINT 3-9 AND loc16,#16bitSigned 6-50 ANDB AX, #8bit 6-51 architectural overview 1-1 architecture differences between the C27x and the C28x E-1 architecture differences between the C2xLP and the C28x C-1 arithmetic logic unit (ALU) 1-5 arithmetic shift right 6-53 ARP C-9 ARx registers D-14 ASP 6-52 ASR AX,#1016 6-53 ASR AX,T 6-54 ASR64 ACC:P.#1..16 6-55 ASR64 ACC:P,T 6-56 ASRL ACC,T 6-57 atomic arithmetic logic unit (ALU) 2-2 Auxiliary registers C-4 auxiliary registers AR0-AR5, XAR6, XAR7 2-12 pointer 2-34

B

B 16bitOffset,COND 6-58 B0 Memory Map C-14 background code 7-6 BANZ E-4 BANZ 16bitOffset,ARn-- 6-59 BAR 16bitOffset,ARn,ARm,EQ 6-60 barrel shifter 1-5 benchmark counter 7-20 BF 16bitOffset,COND 6-61 bits auxiliary register pointer (ARP) 2-34 carry (C) 2-25

debug enable mask (DBGM) 2-37 debug interrupt enable register (DBGIER) 3-10 emulation access enable (EALLOW) 2-35 IDLE status (IDLESTAT) 2-35 interrupt enable register (IER) 3-9 interrupt flag register (IFR) 3-7 interrupt global mask (INTM) 2-37 loop instruction status (LOOP) 2-35 negative flag (N) 2-24 overflow counter (OVC) 2-16 overflow flag (V) 2-21 overflow mode (OVM) 2-32 PAGE0 addressing mode configuration 2-36 product shift mode (PM) 2-19 sign-extension mode 2-32 stack pointer alignment (SPA) 2-36 test/control flag (TC) 2-30 vector map (VMAP) 2-36 zero flag (Z) 2-25 block diagram of the CPU, figure 2-3 branch 6-58 instructions introduced 2-39 break event 7-6 break events 7-7 breakpoints 7-19 caution about time-critical ISRs 7-11 Building a C27x Object File From C27x Source, figure F-8 buses data-/program-write data 1-9 data-read address 1-9 data-read data 1-9 data-write address 1-9 program address 1-9 program-read data 1-9 special operations 1-10 summary table 1-10

С

C bit 2-25 C27MAP 6-62 C27OBJ 6-63 C27x Compatible Mapping of ABlocks M0 and M1, figure F-7 C27x object mode F-8 C28ADDR 6-64 C28MAP 6-65 C28OBJ 6-66

C28x and C2xLP Flags, table E-2 C28x features C-2 C28x Product Mode Shifter, table C-8 C28x Status Register ST0 C-7 C28x Status Register ST1 C-7 C2xLP D-1 C2xLP and C28x architectural differences C-1 C2xLP and C28x Differences in Instructions and Registers, table D-13 C2xLP and C28x Differences in Interrupts, table D-10 C2xLp and C28x Differences in Memory Maps, table D-12 C2xLP and C28x Differences in Status Registers, table D-11 C2xLP Instructions and C28x Equivalent Instructions, table E-3 C2xLP Product Mode Shifter C-8 C2xLP Status Register ST0 C-7 C2xLP Status Register ST1 C-7 calls 2-39 Carry bit (C) C-10 carry bit (C) 2-25 caution, breakpoints within time-critical interrupt service routines 7-11 central processing unit (CPU) 1-4, 2-2 reset 3-23 in real-time mode debug-halt state 7-9 checksum computation B-6 circular addressing modes 5-21 clear the AMODE bit 6-67 clear the OBJMODE bit 6-63 CLRC AMODE 6-67 CLRC M0M1MAP 6-68 CLRC OBJMODE 6-69 CLRC OVC 6-70 CLRC XF 6-71 CLRC mode 6-72 CMP AX, loc16 6-74 CMP loc16,#16bitSigned 6-75 CMP64 ACC:P 6-77 CMPB AX, #8bit 6-79 CMPL ACC,loc32 6-80 CMPL ACC,P < PM 6-81 CMPR 0 6-82

code clear IFR D-8 conversion from C2xLP D-8 IER/IFR D-7 interrupt D-8 migration reference tables D-10 code examples D-7 Code for a Full Context Save/Restore for C28x vs C27x, figure F-6 code submission B-4 compare 6-74 compatibility 1-2 compiler 5-7 computation of checksum B-6 core 1-2 components 1-4 diagram 1-4 count sign bits 6-83 counters 7-20 CPU 1-4, 2-2 See also central processing unit reset 3-23 in real-time mode debug-halt state 7-9 CPU registers 2-4 CSB ACC 6-83 CSM passwords B-5 custom ROM codes B-1

D

data buses 1-9 data log interrupt (DLOGINT) 3-6, 7-27 vector 3-5 data logging 1-5, 7-23 accessing emulation registers 7-26 creating a transfer buffer 7-23 examples 7-28 interrupt (DLOGINT) 3-6, 7-27 interrupt vector 3-5 with end address 7-29 with word counter 7-28 data logging end-address control register 7-26 data memory C-14 data move contents 6-89 data page pointer C-5 data page pointer (DP) 2-10 data space, address map 1-8
data-/program-write data bus (DWDB) 1-9 data-read address bus (DRAB) 1-9 data-read data bus (DRDB) 1-9 data-write address bus (DWAB) 1-9 DBGIER A-2 DBGM F-4 debug enable mask bit C-9 DBGSTAT register 7-15 debug enable mask bit (DBGM) 2-37 event 7-6 execution control modes 7-7 halt state 7-6 interface 7-3 sharing resources 7-30 terminology 7-6 debug enable mask bit (DBGM) 2-37 role in accesses during emulation 7-16 set during interrupt handling 3-15, 3-20 debug interrupt enable register A-2 debug interrupt enable register (DBGIER) 3-6, 3-8, 3-10, 7-9 quick reference figure A-9 Debug interrupt-enable register C-4 debug status register (DBGSTAT) 7-15 debug-and-test direct memory access 1-5 debug-and-test direct memory access (DT-DMA) mechanism 7-16 debug-halt state 7-7, 7-9 DEC loc16 6-84 decoupled pipeline segments 4-4 decrement by 1 6-84 development interface 7-2 diagnostic features for emulation 7-31 diagrams CPU 2-3 memory map 1-7 multiplication 2-42, 2-43 pipeline activity 4-8 pipeline conflict 4-13, 4-14 relationship between pipeline and address counters 4-6 shift operations 2-45 T320C28x DSP core 1-4 DINT 6-85 Direct Addressing Mode 5-2, 5-8

direct addressing mode C-5 Direct Addressing Mode Mapping, figure C-6 direct addressing mode on the C2xLP C-5 direct memory access mechanism for emulation 7-16 disable write access to protected registers 6-91 discontinuity delay 4-11 DMA control register 7-25 DMA ID register 7-25 DMA registers (data logging) 7-25 DMAC ACC:P,loc32,*XAR7 6-86 DMOV loc16 6-89 DP 2-10 DT-DMA 1-5 DT-DMA mechanism 7-16 DT-DMA request process, figure 7-17 dual multiply and accumulate 6-86

EALLOW 6-90, C-9 EALLOW bit 2-35 EALLOW instruction, use in data logging 7-23, 7-26 EDIS 6-91 EDIS instruction, use in data logging 7-24, 7-27 EINT 6-92 EMU0/1, signals 7-4 EMU0/1 signals 7-4 emulation data logging 7-23 disabled 7-5 enabled 7-5 features 1-5, 7-2 logic 1-4, 1-5, 7-15 Emulation access enable bit C-9 emulation access enable bit (EALLOW) 2-35 emulation signals 1-6 enable maskable interrupts 6-92 enable write access to protected space 6-90 end address register (data logging) 7-26 ESTOP0 6-93 ESTOP1 6-94 event counter 7-20 events break 7-6

Index-4

debug 7-6 examples code D-7 data logging with end address 7-29 data logging with word counter 7-28 execution control modes real-time mode 7-9 stop mode 7-7

F

fast function call 6-95 FC (fetch counter) 4-5 fetch counter (FC) 4-5 FFC XAR7,22bit 6-95 find the maximum 6-149 find the minimum 6-153 flags, interrupt flag register (IFR) 3-7 FLIP AX 6-96 flip order of bits in AX register 6-96 flow chart of recommended migration steps, figure D-4 flow charts handling DT-DMA request 7-17 interrupt initiated by the TRAP instruction 3-18 interrupt operation, maskable interrupts 3-12 foreground code 7-6 full context save D-8 Full Context Save/Restore, figure F-5 Full Context Save/Restore Comparison, table D-9

G

global space C-14 GREG register D-14

Η

hardware reset 3-23 hardware reset interrupt 3-17 header, dimensions, 14-pin 7-3 high-impedance mode 7-5

I/O space C-14 IACK #16bit 6-97 IC (instruction counter) 4-5 IDLE 6-98 IDLE status bit (IDLESTAT) 2-35 **IDLESTAT C-9** IDLESTAT bit 2-35 IEEE 1149.1 (JTAG) signals 7-3 IER A-2 IFR A-2 illegal-instruction trap 3-17, 3-22 IMACL P,loc32,*XAR7 6-100 improvements over the C2xLP CPU C-2 IMPYAL P,XT,loc32 6-103 IMPYL ACC, XT, loc32 6-105 IMPYL P,XT,loc32 6-106 IMPYSL P,XT,loc32 6-107 IMPYXUL P,XT,loc32 6-109 IN loc16,*(PA) 6-111 INC loc16 6-113 increment by 1 6-113 Indirect Addressing Mode 5-2, 5-10 individually accessible portions of the accumulator 2-7 instrucitons, MAX AX,loc16 (find the maximum) 6-149 instruction TBIT 6-359 XB 6-370 instruction counter (IC) 4-5 Instruction Syntax Change, table F-12 instruction-fetch mechanism 4-4 instruction-not-available condition 4-10 instructions ABORTI (abort interrupt) 7-15 ABS ACC (Absolute Value of Accumulator) 6-19 ADD (add constant) 6-29 ADD AX (Add value to AX) 6-27 ADDACC (Add value to accumulator) 6-22 ADDB XARn, #7bit 6-33 ADDUACC (add unsigned value to accumulator) 6-39 ADRK (add to current register) 6-42

ASP (align stack pointer) 6-52 ASRAX (arithmetic shift right) 6-53 B (branch) 6-58 C27MAP (set the M0M1MAP bit) 6-62 C27OBJ (clear the OBJMODE bit) 6-63 CLRCAMODE (clear the AMODE bit) 6-67 CMP AX (compare) 6-74 conditional 2-39 CSBACC (count sign bits) 6-83 DEC (decrement) 6-84 DMA ACC (dual multiply and accumulate) 6-86 DMOV (data move contents) 6-89 EALLOW (enable write access to protected space 6-90 EDIS (disable write access to protected registers) 6-91 EINT (enable maskable interrupts) 6-92 FFC XAR7,22bit (fast function call) 6-95 FLIP AX (flip order of bits in AX register) 6-96 IACK,#16bit (interrupt acknowledge 6-97 INCloc16 (increment by 1) 6-113 INTR (software interrupt) 3-17 LB *XAR7 (long indirect branch) 6-119 LC *XAR7 (long indirect call) 6-121 LCR #22bit (long call using RPC) 6-123 LOOPNZ loc16,#16bit (loop while not zero) 6-125 LPADDR (set the AMODE bit) 6-129 LRET (long return) 6-130 LSLACC,#1..16 (logical shift left) 6-133 LSR AX,#1..16 (logical shift right) 6-140 MAC (multiply and accumulate, preload T) 4-16 MIN AX,loc16 (find the minimum) 6-153 MOV AR6/7,loc16 (load auxiliary register) 6-160 MOV AX,loc16 (load AX) 6-161 MOV DP,#10bit (load data page pointer) 6-162 MOV IER, loc16 (load the interrupt enable register) 6-163 MOV loc16, #16bit (save 16-bit constant) 6-164 MOV loc16, OVC (store the overflow counter) 6-173 MOV OVC,loc16 (load the overflow counter) 6-176 MOV*(0:16bit),loc16 (move value) 6-156 OR ACC,loc16 (bitwise OR) 6-257 OUT *(PA),loc16 (output data to port) 6-265

POP ACC (pop top of stack to accumulator) 6-267 PREAD (read from program memory) 4-16 PREAD loc16,*XAR7 (read from program memory) 6-282 PUSH ACC (push accumulator onto stack) 6-284 PWRITE (write to program memory) 4-16 ROL ACC (rotate accumulator left) 6-310 ROR ACC (rotate accumulator right) 6-311 SAT ACC (saturate accumulator) 6-313 SUBR loc16,AX 6-354 TRAP (software trap) 3-17 instructions , PWRITE *XAR7,loc16 (write to program memory) 6-299 interface, memory 1-9 interrupt, signals 1-6 interrupt acknowledge 6-97 interrupt-control registers (IFR, IER, DBGI-ER) 2-14 interrupt enable D-10 Interrupt enable register C-4 interrupt enable register D-10 interrupt enable register (IER) 3-6, 3-8, 7-9 quick reference figure A-8 Interrupt flag register C-4 interrupt flag register A-2, D-10 interrupt flag register (IFR) 3-7 quick reference figure A-7 Interrupt global mask bit (INTM) C-10 interrupt global mask bit (INTM) 2-37, 3-6, 7-9 interrupt handling in real-time mode 7-9 interrupt handling in stop mode 7-7 interrupt instructions AND IER 3-8 AND IFR 3-7 INTR 3-8, 3-17 OR IER 3-8 OR IFR 3-7 POP DBGIER 3-10 PUSH DBGIER 3-10 TRAP 3-17 interrupt service routine (ISR) 3-4 caution about breakpoints 7-11 interrupt vectors 1-7 interrupts 2-39, 3-1 aborting 7-15

control registers (IFR, IER, DBGIER) 2-14 data log interrupt (DLOGINT) 3-6, 7-27 effect on instructions in pipeline 4-4 general purpose 3-6 handling information by emulation mode and state 7-13 INT1-INT14 3-6 maskable 3-6 definition 3-2 flow chart of operation 3-12 NMI 3-21 nonmaskable 3-17 definition 3-2 operation overview 3-2 real-time mode 7-9 standard 3-11 stop mode 7-7 overview 3-2 real-time operating system interrupt (RTO-SINT) 3-6 special cases, clearing IFR flag bit after TRAP instruction 3-7, 3-8 time-critical 7-6 serviced in real-time mode 7-9 vectors 3-4 INTM C-11 INTR INTx 6-114 INTR instruction 3-17, D-10 IRET 6-116 IRET instruction 7-15, F-5

J

JTAG, signals 7-3 JTAG header to interface a target to the scan controller, figure 7-3 JTAG port 7-1

L

LACL dma D-15 LB *XAR7 6-119 LB 22bit 6-120 LC *XAR7 6-121 LC 22bit 6-122 LCR #22bit 6-123 LCR *XARn 6-124 load auxiliary register 6-160 load AX 6-161 load data page pointer 6-162 load the interrupt enable register 6-163 load the overflow counter 6-176 loc16 5-2 loc32 5-2 logical shift left 6-133 logical shift right 6-140 long call using RPC 6-123 long indirect branch 6-119 long indirect call 6-121 long return 6-130 LOOP C-9 LOOP bit 2-35 Loop instruction status bit C-9 loop instruction status bit (LOOP) 2-35 loop while not zero 6-125 LOOPNZ loc16,#16bit 6-125 LOOPZ loc16,#16bit 6-127 LPADDR 6-129 LRET 6-130 LRETE 6-131 **LRETER 6-132** LSL ACC,#1..16 6-133 LSL ACC,T 6-134 LSL AX,#1016 6-135 LSL AX,T 6-136 LSL64 ACC:P,#1..16 6-137 LSL64 ACC:P,T 6-138 LSLL ACC,T 6-139 LSR AX,#1016 6-140 LSR AX,T 6-141 LSR64 ACC:P,#1..16 6-142 LSR64 ACC:P,T 6-143 LSRL ACC,T 6-144

Μ

M0 M1 map bit C-9 M0M1MAP C-9, F-4 MAC P,loc16,0:pma 6-145 MAC P, loc16, *XAR7 6-147 Mapping of memory blocks B0 and B1 on C27 F-7 maskable interrupts 3-6 definition 3-2 flow chart of operation 3-12 MAX AX, loc16 6-149 MAXCUL P,loc32 6-150 MAXL ACC,loc32 6-152 memory 1-9 address map 1-8 interface 1-9 map 1-7 reserved addresses 1-8 Memory Map C-13 memory map C-12, F-2 memory map diagram 1-7 Memory space C-12 memory wrappers 1-11 migration 1-2 migration flow D-3 migration guidelines D-1 MIN AX, loc16 6-153 MINCUL P,loc32 6-154 MINL ACC, loc32 6-155 mixing of C2xLP code and C28x code segments D-6 modes high-impedance 7-5 nonpreemptive 7-16 normal with emulation disabled 7-5 normal with emulation enabled 7-5 preemptive 7-16 real-time 7-7, 7-9 slave 7-5 stop 7-7 MOV *(0:16bit),loc16 6-156 MOV AX, loc16 6-161 MOV ACC,#16bit<#0..15 6-157 MOV ACC,loc16 < T 6-158 MOV ACC,loc16<#0..16 6-159 MOV AR6, loc16 6-160 MOV DP, #10bit 6-162 MOV IER,loc16 6-163 MOV loc16, #0 6-166 MOV loc16, #16bit 6-164 MOV loc16, *(0:16bit) 6-165 MOV loc16, AX 6-169

MOV loc16, AX, COND 6-170 MOV loc16,IER 6-172 MOV loc16,OVC 6-173 MOV loc16,P 6-174 MOV OVC, loc16 6-176 MOV PH, loc16 6-177 MOV PL, loc16 6-178 MOV PM, AX 6-179 MOV T, loc16 6-180 MOV TL, #0 6-181 MOV loc16,ARn 6-168 MOV XARn, PC 6-182 MOV, loc16,T 6-175 MOVA, T,loc16 6-183 MOVAD T, loc16 6-185 MOVB ACC,#8bit 6-187 MOVB AR6/7, #8bit 6-188 MOVB AX.LSB, loc16 6-190 MOVB AX.MSB, loc16 6-192 MOVB AX, #8bit 6-189 MOVB loc16, AX.LSB 6-196 MOVB loc16, AX.MSB 6-198 MOVB loc16,#8bit,COND 6-194 MOVB XARn, #8bit 6-200 MOVDL XT,loc16 6-201 move value 6-156 MOVH loc16, P 6-203 MOVH loc16, ACC, < #1..8 6-167, 6-202 MOVL ACC,loc32 6-204 MOVL ACC,P < PM 6-205 MOVL loc32, ACC 6-206 MOVL loc32, XAR0 6-210 MOVL loc32, ACC, COND 6-207 MOVL loc32,P 6-209 MOVL loc32,XT 6-211 MOVL P,ACC 6-212 MOVL P,loc32 6-213 MOVL XAR0, loc32 6-214 MOVL XARn, #22bit 6-215 MOVL XT, loc32 6-216 MOVP T,loc16 6-217 MOVS, T,loc16 6-218 MOVU ACC,loc16 6-220 MOVU loc16,OVC 6-221

MOVU OVC.loc16 6-222 MOVW DP, #16bit 6-223 MOVX TL,loc16 6-224 MOVZ AR005, loc16 6-225 MOVZ DP. #10bit 6-226 MPY ACC, loc16,#16bit 6-227 MPY ACC, T, loc16 6-228 MPY P,loc16,#16bit 6-229 MPY P,T,loc16 6-230 MPYA P,loc16,#16bit 6-231 MPYA P,T,loc16 6-233 MPYB ACC,T,#8bit 6-235 MPYB P,T,#8bit 6-236 MPYS P,T,loc16 6-237 MPYU ACC, T, loc16 6-240 MPYU P.T.loc16 6-239 MPYXU ACC, T, loc16 6-241 MPYXU P,T,loc16 6-242 Multiplicand register C-4 multiplicand register (T) 2-8 multiplier, operation 2-41

Ν

N bit 2-24 NASP 6-243 NEG ACC 6-244 NEG AX 6-245 NEG64 ACC:P 6-246 Negative flag C-8 negative flag (N) 2-24 NEGTC ACC 6-248 NMI Instruction D-10 NMI interrupt 3-21 NMI pin 3-21 nonmaskable interrupts 3-17 definition 3-2 nonpreemptive mode 7-16 NOP {*ind}{,ARPn} 6-250 NORM ACC, *ind 6-251 NORM ACC.XARn++ 6-253 normal mode 7-5 NOT ACC 6-255 NOT AX 6-256

0

OBJMODE C-9, F-4, F-9 OBJMODE bit 1-2 operating modes, selecting by using TRST, EMU0, and EMU1 7-5 operations multiply 2-41 shift 2-44 special bus 1-10 stack 2-12 OR ACC, loc16 6-257 OR ACC,#16bit < #0..15 6-258 OR AX, loc16 6-259 OR loc16, AX 6-263 OR IER,#16bit 6-260 OR IFR,#16bit 6-261 OR loc16,#16bit 6-262 ORB AX, #8bit 6-264 OUT *(PA),loc16 6-265 output data to port 6-265 OVC, overflow counter C-9 OVC (overflow counter) 2-16 overflow counter (OVC) 2-16 Overflow flag C-8 overflow flag (V) 2-21 Overflow mode (OVM) C-10 overflow mode bit (OVM) 2-32 OVM C-11 OVM bit 2-32

P register 2-9 PAGE0 addressing mode configuration bit C-9 PAGE0 bit 2-36 PC (program counter) 2-14, 4-5 phases of pipeline 4-2 pipeline 2-40 decoupled segments 4-4 freezes in activity 4-10 instruction-fetch mechanism 4-4 operations not protected by 4-16 phases 4-2 protection 4-12 visualizing activity 4-7

wait states 4-10 pipeline phases 4-2 PM bits 2-19 POP ACC 6-267 POP AR1:AR0 6-268 POP AR1H:AR0H 6-269 POP AR3:AR2 6-268 POP AR5:AR4 6-268 POP DBGIER 6-270 POP DP 6-271 POP DP:ST1 6-272 POP IFR 6-273 POP loc16 6-274 POP P 6-275 POP RPC 6-276 POP ST0 6-277 POP ST1 6-278 POP T:ST0 6-279 pop top of stack to accumulator 6-267 POP XAR0 6-280 POP XAR1 6-280 POP XAR2 6-280 POP XAR3 6-280 POP XAR4 6-280 POP XAR5 6-280 POP XAR6 6-280 POP XAR7 6-280 POP XT 6-281 PREAD loc16,*XAR7 6-282 preemptive mode 7-16 process for handling a DT-DMA request, figure 7-17 Product Mode Shifter C-8 Product register C-4 product register (P) 2-9 Product shift mode C-8 product shift mode bits (PM) 2-19 program address bus (PAB) 1-9, 4-4 Program counter C-4 program counter D-14 program counter (PC) 2-14, 4-5 program flow 2-39 Program space C-12 program space, address map 1-8

program-space read and write 1-10 program-address counters 4-5 program-read data bus (PRDB) 1-9 PUSH ACC 6-284 push accumulator onto stack 6-284 PUSH AR1:AR0 6-285 PUSH AR1H:AR0H 6-286 PUSH AR3:AR2 6-285 PUSH AR5:AR4 6-285 PUSH DBGIER 6-287 PUSH DP 6-288 PUSH DP:ST1 6-289 PUSH IFR 6-290 PUSH loc16 6-291 PUSH P 6-292 PUSH RPC 6-293 PUSH ST0 6-294 PUSH ST1 6-295 PUSH T:ST0 6-296 PUSH XAR0 6-297 PUSH XAR1 6-297 PUSH XAR2 6-297 PUSH XAR3 6-297 PUSH XAR4 6-297 PUSH XAR5 6-297 PUSH XAR6 6-297 PUSH XAR7 6-297 PUSH XT 6-298 PWRITE *XAR7, loc16 6-299

Q

QMACL P,loc32,*XAR7 6-300 QMACL P,loc32,*XAR7++ 6-300 QMPYAL P,XT,loc32 6-302 QMPYL P,XT,loc32 6-304 QMPYL ACC,XT,loc32 6-304 QMPYSL P,XT,loc32 6-306 QMPYUL P,XT,loc32 6-308 QMPYXUL P,XT,loc32 6-309

R

read from program memory 6-282 reads and writes, unprotected 4-16

Index-10

real-time mode 7-7, 7-9 figure of execution states 7-10 real-time mode versus stop mode, figure 7-12 real-time operating system interrupt (RTOSINT) 3-6, 7-14 Register Addressing Mode 5-2 register addressing modes 5-25 register changes C-4 register modifications C-3, F-2 register guick reference A-1 figures A-3 registers accumulator 2-6 ADDRH 7-24 ADDRL 7-24 after reset 3-23 auxiliary registers (XAR0 - XAR7) 2-12 conflicts, protection against 4-13 CPU registers (summary) 2-4 data page pointer (DP) 2-10 debug interrupt enable register (DBGIER) 3-8 DMA control register 7-25 end address register (data logging) 7-26 interrupt-control registers (IFR, IER, DBGIER) 2-14 interrupt enable register (IER) 3-8 interrupt flag register (IFR) 3-7 multiplicand (T) 2-8 product register (P) 2-9 program counter (PC) 2-14 quick reference A-1 quick reference figures A-3 return program counter (RPC) 2-14 stack pointer (SP) 2-11 start address register (data logging) 7-25 status register ST0 2-14, 2-16 status register ST1 2-14, 2-34 T register 2-8 registers after reset 3-23 repeat counter (RPTC) 2-39 repeat instructions D-13 repeatable instructions E-9, F-13 reserved addresses 1-8 Reserved memory C-14 reset 1-3 reset and interrupt signals 1-6 reset conditions C-10

Reset Conditions of Internal Registers, table C-10 reset input signal (RS) 3-23 reset of CPU 3-23 Reset Values of the Status and Control Registers, table A-2 Return Program Counter 2-5 Return program counter C-4 return program counter (RPC) 2-14 returns 2-39 ROL ACC 6-310 ROM code generation flow B-6 ROM codes, submitting custom B-1 ROM layout B-5 ROR ACC 6-311 rotate accumulator left 6-310 RPC (return program counter) 2-14 RPT #8bit 6-312 RPT loc16 6-312 RPTC (repeat counter) 2-39 run state 7-7, 7-10

S

SARAM mapping D-13 SAT ACC 6-313 SAT64 ACC:P 6-314 save 16-bit constant 6-164 SB 8bitOffset,COND 6-316 SBBU ACC,loc16 6-317 SBF 8bitOffset.EQ 6-318 SBF 8bitOffset,NEQ 6-318 SBF 8bitOffset,NTC 6-318 SBF 8bitOffset,TC 6-318 selecting device operating modes 7-5 set the AMODE bit 6-129 set the M0M1MAP bit 6-62 SETC C 6-320 SETC DBGM 6-320 SETC INTM 6-320 SETC OVM 6-320 SETC PAGE0 6-320 SXM 6-320 SETC SETC TC 6-320 SETC VMAP 6-320

SETC MOM1MAP 6-322 SETC mode 6-320 SETC OBJMODE 6-323 SETC XF 6-324 SFR ACC,#1..16 6-325 SFR ACC,T 6-326 shift operations 2-44 shifter 1-5 shifting values in the accumulator 2-8 Sign extension mode (SXM) C-10 signal descriptions, 14-pin header 7-4 signals 1-6 description, 14-pin header 7-4 EMU0 7-5 EMU1 7-5 PD (VCC) 7-3 TCK 7-3 TCK RET 7-3 TDI 7-3 TDO 7-3 TMS 7-3 TRST 7-3, 7-5 sign-extension mode bit (SXM) 2-32 single-instruction state 7-7 slave mode 7-5 software breakpoints 7-7 software interrupts 3-17 SPA bit 2-36 special bus operations 1-11 SPM +1 6-327 SPM +4 6-327 SPM -1 6-327 SPM -2 6-327 SPM -3 6-327 SPM -4 6-327 SPM -5 6-327 SPM -6 6-327 SPM 0 6-327 SQRA loc16 6-329 SQRS loc16 6-331 ST0 A-2 ST0 Register Bits, table F-3 ST1 A-2 ST1 Register Bits, table F-4 stack 2-11

Stack Addressing Mode 5-2, 5-9 Stack Pointer C-4 stack pointer (SP) 2-11 Stack pointer alignment bit C-9 stack pointer alignment bit (SPA) 2-36 Stack space C-14 start address register (data logging) 7-25 status bits ARP 2-34 C 2-25 DBGM 2-37 EALLOW 2-35 **IDLESTAT 2-35** INTM 2-37 LOOP 2-35 N 2-24 OVC 2-16 OVM 2-32 PAGE0 2-36 PM 2-19 SPA 2-36 SXM 2-32 TC 2-30 V 2-21 VMAP 2-36 Z 2-25 status register A-2 Status Register Bits C-11 status register changes C-7 Status Register Comparison Between C2xLP and C28x, figure C-7 Status Registers C-5 status registers ST0 2-14, 2-16 quick reference figure A-4 ST1 2-14, 2-34 quick reference figure A-5 stop mode 7-7 figure of execution states 7-8 stop mode versus real-time mode, figure 7-12 store the overflow counter 6-173 SUB loc16, AX 6-339 SUB ACC,#16bit < #0..15 6-337 SUB ACC,loc16 < #0 6-333 SUB ACC,loc16 < T 6-335 SUB AX, loc16 6-338 SUBB ACC,#8bit 6-340 SUBB XARn, #7bit 6-342

D-1 x D-1

SUBB SP.#7bit 6-341 SUBBL ACC, loc32 6-343 SUBCU ACC, loc16 6-345 SUBCU instruction F-14 SUBCUL ACC,loc32 6-347 SUBL ACC, loc32 6-350 SUBL ACC,P < PM 6-351 SUBL loc32, ACC 6-353 submitting ROM codes to TI B-1 SUBR loc16,AX 6-354 SUBRL loc32, ACC 6-355 SUBU ACC, loc16 6-356 SUBUL ACC, loc32 6-357 SUBUL P,loc32 6-358 suspend program execution 7-7 SXM bit 2-32 syntax change increment/decrement D-15 repeat instructions D-15 shift D-15



T register 2-8 T320C28x core 1-2 TBIT loc16,#16bit 6-359 TBIT loc16.T 6-360 TC bit 2-30 TCK signal 7-4 TCLR loc16,#bit 6-361 TDI signal 7-4 terminology, debug 7-6 test, sharing resources 7-30 TEST ACC 6-362 test clock return signal (TCK RET) 7-3 Test/control flag (TC) C-10 test/control flag bit (TC) 2-30 testing and debugging, signals 1-6 TI internal testing B-5 time-critical interrupts definition 7-6 serviced in real-time mode 7-9 TMS signal 7-4 TMS320C20x D-1

TMS320C24x D-1 TMS320C24xx D-1 TRAP #VectorNumber 6-363 TRAP instruction 3-18, D-10 TRST signal 7-4, 7-5 TSET loc16,#16bit 6-365 types of signals 1-6

U

unprotected program-space reads and writes 4-16 UOUT *(PA),loc16 6-366

V

V bit 2-21 Vector map bit C-9 vector map bit (VMAP) 2-36 Vectors C-12 VMAP F-4

W

wait states, effects on pipeline 4-10 wait-in-reset mode 7-5 watchpoints 7-19 write to program memory 6-299

Χ

XAR6 register 2-12 XARn registers F-9 XB *AL 6-368 XB pma,COND 6-370 XB pma,*,ARPn 6-369 XBANZ pma,* 6-372 XBANZ pma,* 6-372 XBANZ pma,*++ 6-372 XBANZ pma,*++,ARPn 6-372 XBANZ pma,*-- 6-372 XBANZ pma,*--,ARPn 6-372 XBANZ pma,*0++ 6-372 XBANZ pma,*0++ 6-372 XBANZ pma,*0-- 6-372 XBANZ pma,*0-- 6-372 XBANZ pma,*0-- 6-372 XCALL *AL 6-374 XCALL pma,*,ARPn 6-375 XCALL pma,COND 6-376 XF F-4 XF pin status bit C-9 XMAC P,loc16,*(pma) 6-378 XMACD P,loc16,*(pma) 6-380 XOR AX, loc16 6-384 XOR ACC,#16bit <#0..15 6-383 XOR ACC,loc16 6-382 XOR loc16, AX 6-385 XOR loc16,#16bit 6-386 XORB AX, #8bit 6-387 XPREAD loc16,*(pma) 6-388 XPREAD loc16,*AL 6-389 XPWRITE *AL,loc16 6-390 XRET 6-391 XRETC COND 6-392



ZAP OVC 6-395 ZAPA 6-396 Zero flag C-8 zero flag bit (Z) 2-25