TMS320C54x DSP Reference Set

Volume 2: Mnemonic Instruction Set

Literature Number: SPRU172C
March 2001

TEXAS INSTRUMENTS

PRINTED WITH SOY INK™

Printed on Recycled Paper

IMPORTANT NOTICE

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

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

Customers are responsible for their applications using TI components.

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

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2001, Texas Instruments Incorporated

Preface

Read This First

About This Manual

The TMS320C54x™ DSP is a fixed-point digital signal processor (DSP) in the TMS320™ DSP family and it can use either of two forms of the instruction set: a mnemonic form or an algebraic form. This book is a reference for the mnemonic form of the instruction set. It contains information about the instructions used for all types of operations (arithmetic, logical, load and store, conditional, and program control), the nomenclature used in describing the instruction operation, and supplemental information you may need, such as interrupt priorities and locations. For information about the algebraic form of the instruction set, see *TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set*, literature number SPRU179.

How to Use This Manual

The following table summarizes the C54x™ DSP information contained in this book:

If you are looking for information about:	Turn to:
Arithmetic operations	Chapter 2, *Instruction Set Summary*
Conditions for conditional instructions	Appendix A, *Condition Codes*
Control register layout	Appendix B, *CPU Status and Control Registers*
Example description of instruction	Chapter 1, *Symbols and Abbreviations*
Individual instruction descriptions	Chapter 4, *Assembly Language Instructions*
Instruction set abbreviations	Chapter 1, *Symbols and Abbreviations*
Instruction set classes	Chapter 3, *Instruction Classes and Cycles*

If you are looking for information about:	Turn to:
Instruction set symbols	Chapter 1, *Symbols and Abbreviations*
Load and store operations	Chapter 2, *Instruction Set Summary*
Logical operations	Chapter 2, *Instruction Set Summary*
Program control operations	Chapter 2, *Instruction Set Summary*
Status register layout	Appendix B, *CPU Status and Control Registers*
Summary of instructions	Chapter 2, *Instruction Set Summary*

Notational Conventions

This book uses the following conventions.

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

  Here is a segment of a program listing:

  ```
  LMS *AR3+, *AR4+
  ```

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

  LMS *Xmem*, *Ymem*

  LMS is the instruction, and it has two parameters, *Xmem* and *Ymem*. When you use **LMS**, the parameters should be actual dual data-memory operand values. A comma and a space (optional) must separate the two values.

- The term OR is used in the assembly language instructions to denote a Boolean operation. The term or is used to indicate selection. Here is an example of an instruction with OR and or:

  lk OR (src) → src or [dst]

  This instruction ORs the value of lk with the contents of src. Then, it stores the result in src or dst, depending on the syntax of the instruction.

- Square brackets, [and], identify an optional parameter. If you use an optional parameter, specify the information within the brackets; do not type the brackets themselves.

Related Documentation From Texas Instruments

The following books describe the TMS320C54x™ DSP and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number. Many of these documents are located on the internet at http://www.ti.com.

TMS320C54x DSP Reference Set, Volume 1: CPU (literature number SPRU131) describes the TMS320C54x™ 16-bit fixed-point general-purpose digital signal processors. Covered are its architecture, internal register structure, data and program addressing, and the instruction pipeline. Also includes development support information, parts lists, and design considerations for using the XDS510™ emulator.

TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set (literature number SPRU172) describes the TMS320C54x™ digital signal processor mnemonic instructions individually. Also includes a summary of instruction set classes and cycles.

TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set (literature number SPRU179) describes the TMS320C54x™ digital signal processor algebraic instructions individually. Also includes a summary of instruction set classes and cycles.

TMS320C54x DSP Reference Set, Volume 4: Applications Guide (literature number SPRU173) describes software and hardware applications for the TMS320C54x™ digital signal processor. Also includes development support information, parts lists, and design considerations for using the XDS510™ emulator.

TMS320C54x DSP Reference Set, Volume 5: Enhanced Peripherals (literature number SPRU302) describes the enhanced peripherals available on the TMS320C54x™ digital signal processors. Includes the multichannel buffered serial ports (McBSPs), direct memory access (DMA) controller, interprocessor communications, and the HPI-8 and HPI-16 host port interfaces.

TMS320C54x DSP Family Functional Overview (literature number SPRU307) provides a functional overview of the devices included in the TMS320C54x™ DSP generation of digital signal processors. Included are descriptions of the CPU architecture, bus structure, memory structure, on-chip peripherals, and instruction set.

TMS320C54x DSKplus User's Guide (literature number SPRU191) describes the TMS320C54x™ digital signal processor starter kit (DSK), which allows you to execute custom TMS320C54x DSP code in real time and debug it line by line. Covered are installation procedures, a description of the debugger and the assembler, customized applications, and initialization routines.

TMS320C54x Code Composer Studio Tutorial (literature number SPRU327) introduces the Code Composer Studio integrated development environment and software tools for the TMS320C54x.

Code Composer User's Guide (literature number SPRU328) explains how to use the Code Composer development environment to build and debug embedded real-time DSP applications.

TMS320C54x Assembly Language Tools User's Guide (literature number SPRU102) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C54x™ generation of devices.

TMS320C54x Optimizing C Compiler User's Guide (literature number SPRU103) describes the TMS320C54x™ C compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for the TMS320C54x generation of devices.

TMS320C54x Simulator Getting Started (literature number SPRU137) describes how to install the TMS320C54x™ simulator and the C source debugger for the TMS320C54x DSP. The installation for MS-DOS™, PC-DOS™, SunOS™, Solaris™, and HP-UX™ systems is covered.

TMS320C54x Evaluation Module Technical Reference (literature number SPRU135) describes the TMS320C54x™ evaluation module, its features, design details and external interfaces.

TMS320C54x Code Generation Tools Getting Started Guide (literature number SPRU147) describes how to install the TMS320C54x™ assembly language tools and the C compiler for the TMS320C54x devices. The installation for MS-DOS™, OS/2™, SunOS™, Solaris™, and HP-UX™ 9.0x systems is covered.

TMS320C5xx C Source Debugger User's Guide (literature number SPRU099) tells you how to invoke the TMS320C54x™ emulator, evaluation module, and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS320C54x Simulator Addendum (literature number SPRU170) tells you how to define and use a memory map to simulate ports for the TMS320C54x™ DSP. This addendum to the *TMS320C5xx C Source Debugger User's Guide* discusses standard serial ports, buffered serial ports, and time division multiplexed (TDM) serial ports.

Setting Up TMS320 DSP Interrupts in C Application Report (literature number SPRA036) describes methods of setting up interrupts for the TMS320™ DSP family of processors in C programming language. Sample code segments are provided, along with complete examples of how to set up interrupt vectors.

TMS320VC5402 and TMS320UC5402 Bootloader (literature number SPRA618) describes the features and operation of the TMS320VC5402 and TMS320UC5402 bootloader. Also discussed is the contents of the on-chip ROM.

TMS320C548/C549 Bootloader Technical Reference (literature number SPRU288) describes the process the bootloader uses to transfer user code from an external source to the program memory at power up. (Presently available only on the internet.)

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

Trademarks

TMS320, TMS320C2x, TMS320C20x, TMS320C24x, TMS320C5x, TMS320C54x, C54x, 320 Hotline On-line, Micro Star, TI, XDS510, and XDS510WS are trademarks of Texas Instruments.

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

MS-DOS and Windows are trademarks of Microsoft Corporation.

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

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

Solaris and SunOS are trademarks of Sun Microsystems, Inc.

SPARC is a trademark of SPARC International, Inc., but licensed exclusively to Sun Microsystems, Inc.

Contents

Figures

Tables

Chapter 1

Symbols and Abbreviations

This chapter lists and defines the symbols and abbreviations used in the instruction set summary and in the individual instruction descriptions. It also provides an example description of an instruction.

Topic **Page**

1.1 Instruction Set Symbols and Abbreviations

Table 1–1 through Table 1–4 list the symbols and abbreviations used in the instruction set summary (Chapter 2) and in the individual instruction descriptions (Chapter 4).

Table 1–1. Instruction Set Symbols and Abbreviations

Symbol	Meaning
A	Accumulator A
ALU	Arithmetic logic unit
AR	Auxiliary register, general usage
ARx	Designates a specific auxiliary register (0 ≤ x ≤ 7)
ARP	Auxiliary register pointer field in ST0; this 3-bit field points to the current auxiliary register (AR).
ASM	5-bit accumulator shift mode field in ST1 (–16 ≤ ASM ≤ 15)
B	Accumulator B
BRAF	Block-repeat active flag in ST1
BRC	Block-repeat counter
BITC	4-bit value that determines which bit of a designated data memory value is tested by the test bit instruction (0 ≤ BITC ≤ 15)
C16	Dual 16-bit/double-precision arithmetic mode bit in ST1
C	Carry bit in ST0
CC	2-bit condition code (0 ≤ CC ≤ 3)
CMPT	Compatibility mode bit in ST1
CPL	Compiler mode bit in ST1
cond	An operand representing a condition used by instructions that execute conditionally
[D]	Delay option
DAB	D address bus
DAR	DAB address register
dmad	16-bit immediate data-memory address (0 ≤ dmad ≤ 65 535)
Dmem	Data-memory operand

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

Symbol	Meaning
DP	9-bit data-memory page pointer field in ST0 ($0 \leq DP \leq 511$)
dst	Destination accumulator (A or B)
dst_	Opposite destination accumulator: If dst = A, then dst_ = B If dst = B, then dst_ = A
EAB	E address bus
EAR	EAB address register
extpmad	23-bit immediate program-memory address
FRCT	Fractional mode bit in ST1
hi(A)	High part of accumulator A (bits 31–16)
HM	Hold mode bit in ST1
IFR	Interrupt flag register
INTM	Interrupt mode bit in ST1
K	Short-immediate value of less than 9 bits
k3	3-bit immediate value ($0 \leq k3 \leq 7$)
k5	5-bit immediate value ($-16 \leq k5 \leq 15$)
k9	9-bit immediate value ($0 \leq k9 \leq 511$)
lk	16-bit long-immediate value
Lmem	32-bit single data-memory operand using long-word addressing
mmr, MMR	Memory-mapped register
MMRx, MMRy	Memory-mapped register, AR0–AR7 or SP
n	Number of words following the XC instruction; n = 1 or 2
N	Designates the status register modified in the RSBX, SSBX, and XC instructions: N = 0 Status register ST0 N = 1 Status register ST1

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

Symbol	Meaning
OVA	Overflow flag for accumulator A in ST0
OVB	Overflow flag for accumulator B in ST0
OVdst	Overflow flag for the destination accumulator (A or B)
OVdst_	Overflow flag for the opposite destination accumulator (A or B)
OVsrc	Overflow flag for the source accumulator (A or B)
OVM	Overflow mode bit in ST1
PA	16-bit port immediate address (0 ≤ PA ≤ 65 535)
PAR	Program address register
PC	Program counter
pmad	16-bit immediate program-memory address (0 ≤ pmad ≤ 65 535)
Pmem	Program-memory operand
PMST	Processor mode status register
prog	Program-memory operand
[R]	Rounding option
RC	Repeat counter
REA	Block-repeat end address register
rnd	Round
RSA	Block-repeat start address register
RTN	Fast-return register used in RETF[D] instruction
SBIT	4-bit value that designates the status register bit number modified in the RSBX, SSBX, and XC instructions (0 ≤ SBIT ≤ 15)
SHFT	4-bit shift value (0 ≤ SHFT ≤ 15)
SHIFT	5-bit shift value (–16 ≤ SHIFT ≤ 15)
Sind	Single data-memory operand using indirect addressing
Smem	16-bit single data-memory operand
SP	Stack pointer
src	Source accumulator (A or B)

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

Symbol	Meaning
ST0, ST1	Status register 0, status register 1
SXM	Sign-extension mode bit in ST1
T	Temporary register
TC	Test/control flag in ST0
TOS	Top of stack
TRN	Transition register
TS	Shift value specified by bits 5–0 of T ($-16 \leq$ TS ≤ 31)
uns	Unsigned
XF	External flag status bit in ST1
XPC	Program counter extension register
Xmem	16-bit dual data-memory operand used in dual-operand instructions and some single-operand instructions
Ymem	16-bit dual data-memory operand used in dual-operand instructions
– – SP	Stack pointer value is decremented by 1
+ + SP	Stack pointer value is incremented by 1
+ + PC	Program counter value is incremented by 1

Table 1–2. Opcode Symbols and Abbreviations

Symbol	Meaning
A	Data-memory address bit
ARX	3-bit value that designates the auxiliary register
BITC	4-bit bit code
CC	2-bit condition code
CCCC CCCC	8-bit condition code
COND	4-bit condition code

Table 1–2. Opcode Symbols and Abbreviations (Continued)

Symbol	Meaning
D	Destination (dst) accumulator bit D = 0 Accumulator A D = 1 Accumulator B
I	Addressing mode bit I = 0 Direct addressing mode I = 1 Indirect addressing mode
K	Short-immediate value of less than 9 bits
MMRX	4-bit value that designates one of nine memory-mapped registers ($0 \leq$ MMRX ≤ 8)
MMRY	4-bit value that designates one of nine memory-mapped registers ($0 \leq$ MMRY ≤ 8)
N	Single bit
NN	2-bit value that determines the type of interrupt
R	Rounding (rnd) option bit R = 0 Execute instruction without rounding R = 1 Round the result
S	Source (src) accumulator bit S = 0 Accumulator A S = 1 Accumulator B
SBIT	4-bit status register bit number
SHFT	4-bit shift value ($0 \leq$ SHFT ≤ 15)
SHIFT	5-bit shift value ($-16 \leq$ SHIFT ≤ 15)
X	Data-memory bit
Y	Data-memory bit
Z	Delay instruction bit Z = 0 Execute instruction without delay Z = 1 Execute instruction with delay

Table 1–3. Instruction Set Notations

<table>
<tr><th>Symbol</th><th>Meaning</th></tr>
<tr><td>Boldface Characters</td><td>Boldface characters in an instruction syntax must be typed as shown.
Example: For the syntax ADD Xmem, Ymem, dst, you can use a variety of values for Xmem and Ymem, but the word ADD must be typed as shown.</td></tr>
<tr><td>italic symbols</td><td>Italic symbols in an instruction syntax represent variables.
Example: For the syntax ADD Xmem, Ymem, dst, you can use a variety of values for Xmem and Ymem.</td></tr>
<tr><td>[x]</td><td>Operands in square brackets are optional.
Example: For the syntax ADD Smem [, SHIFT], src [, dst], you must use a value for Smem and src; however, SHIFT and dst are optional.</td></tr>
<tr><td>#</td><td>Prefix of constants used in immediate addressing. For short- or long-immediate operands, # is used in instructions where there is ambiguity with other addressing modes that use immediate operands. For example:
RPT #15 uses short immediate addressing. It causes the next instruction to be repeated 16 times.
RPT 15 uses direct addressing. The number of times the next instruction repeats is determined by a value stored in memory.
For instructions using immediate operands for which there is no ambiguity, # is accepted by the assembler. For example, RPTZ A, #15 and RPTZ A, 15 are equivalent.</td></tr>
<tr><td>(abc)</td><td>The content of a register or location abc.
Example: (src) means the content of the source accumulator.</td></tr>
<tr><td>x → y</td><td>Value x is assigned to register or location y.
Example: (Smem) → dst means the content of the data-memory value is loaded into the destination accumulator.</td></tr>
<tr><td>r(n–m)</td><td>Bits n through m of register or location r.
Example: src(15–0) means bits 15 through 0 of the source accumulator.</td></tr>
<tr><td><< nn</td><td>Shift of nn bits left (negative or positive)</td></tr>
<tr><td>||</td><td>Parallel instruction</td></tr>
<tr><td>\\</td><td>Rotate left</td></tr>
<tr><td>//</td><td>Rotate right</td></tr>
<tr><td>$\overline{x}$</td><td>Logical inversion (1s complement) of x</td></tr>
<tr><td>| x |</td><td>Absolute value of x</td></tr>
<tr><td>AAh</td><td>Indicates that AA represents a hexadecimal number</td></tr>
</table>

Table 1–4. Operators Used in Instruction Set

Symbols	Operators	Evaluation
+ – ~	Unary plus, minus, 1s complement	Right to left
* / %	Multiplication, division, modulo	Left to right
+ –	Addition, subtraction	Left to right
<< >>	Left shift, right shift	Left to right
< < <	Logical left shift	Left to right
< ≤	Less than, LT or equal	Left to right
> ≥	Greater than, GT or equal	Left to right
≠ !=	Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right

Note: Unary +, –, and * have higher precedence than the binary forms.

1.2 Example Description of Instruction

This example of a typical instruction description is provided to familiarize you with the format of the instruction descriptions and to explain what is described under each heading. Each instruction description in Chapter 4 presents the following information:

- Assembler syntax
- Operands
- Opcode
- Execution
- Status Bits
- Description
- Words
- Cycles
- Classes
- Examples

Each instruction description begins with an assembly syntax expression. Labels may be placed either before the instruction on the same line or on the preceding line in the first column. An optional comment field may conclude the syntax expression. Spaces are required between the fields:

- Label
- Command and operands
- Comment

Syntax

1: **EXAMPLE** *Smem, src*
2: **EXAMPLE** *Smem,* **TS***, src*
3: **EXAMPLE** *Smem,* **16**, *src* [, *dst*]
4: **EXAMPLE** *Smem* [, *SHIFT*]*, src* [*, dst*]

Each instruction description begins with an assembly syntax expression. See Section 1.1 on page 1-2 for definitions of symbols in the syntax.

Operands

Smem:	Single data-memory operand
Xmem, Ymem:	Dual data-memory operands
src, dst:	A (accumulator A)
	B (accumulator B)

$-16 \leq$ SHIFT ≤ 15

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

Opcode

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

The opcode breaks down the various bit fields that make up each instruction. See Section 1.1 on page 1-2 for definitions of symbols in the instruction opcode.

Execution

1: (Smem) + (src) → src
2: (Smem) << (TS) + (src) → src
3: (Smem) << 16 + (src) → dst
4: (Smem) [<< SHIFT] + (src) → dst

The execution section describes the processing that takes place when the instruction is executed. The example executions are numbered to correspond to the numbered syntaxes. See Section 1.1 on page 1-2 for definitions of symbols in the execution.

Status Bits

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

Description

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

Words This field specifies the number of memory words required to store the instruction and its extension words. For instructions operating in single-addressing mode, the number of words given is for all modifiers except for long-offset modifiers, which require one additional word.

Cycles This field specifies the number of cycles required for a given C54x DSP instruction to execute as a single instruction with data accesses in DARAM and program accesses from ROM. Additional details on the number of cycles required for other memory configurations and repeat modes are given in Chapter 3, *Instruction Classes and Cycles*.

Classes This field specifies the instruction class for each syntax of the instruction. See Chapter 3, *Instruction Classes and Cycles*, for a description of each class.

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

Chapter 2

Instruction Set Summary

The TMS320C54x™ DSP instruction set can be divided into four basic types of operations:

- ❑ Arithmetic operations
- ❑ Logical operations
- ❑ Program-control operations
- ❑ Load and store operations

In this chapter, each of the types of operations is divided into smaller groups of instructions with similar functions. With each instruction listing, you will find the best possible numbers for word count and cycle time, and the instruction class. You will also find a page number that directs you to the appropriate place in the instruction set of Chapter 4. Also included is information on repeating a single instruction and a list of nonrepeatable instructions.

Topic **Page**

2.1 Arithmetic Operations

This section summarizes the arithmetic operation instructions. Table 2–1 through Table 2–6 list the instructions within the following functional groups:

- Add instructions (Table 2–1)
- Subtract instructions (Table 2–2 on page 2-3)
- Multiply instructions (Table 2–3 on page 2-4)
- Multiply-accumulate instructions (Table 2–4 on page 2-4)
- Multiply-subtract instructions (Table 2–4 on page 2-4)
- Double (32-bit operand) instructions (Table 2–5 on page 2-6)
- Application-specific instructions (Table 2–6 on page 2-7)

Table 2–1. Add Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ADD *Smem*, *src*	src = src + Smem	1	1	3A, 3B	4-4
ADD *Smem*, TS, *src*	src = src + Smem << TS	1	1	3A, 3B	4-4
ADD *Smem*, 16, *src* [, *dst*]	dst = src + Smem << 16	1	1	3A, 3B	4-4
ADD *Smem* [, *SHIFT*], *src* [, *dst*]	dst = src + Smem << SHIFT	2	2	4A, 4B	4-4
ADD *Xmem*, *SHFT*, *src*	src = src + Xmem << SHFT	1	1	3A	4-4
ADD *Xmem*, *Ymem*, *dst*	dst = Xmem << 16 + Ymem << 16	1	1	7	4-4
ADD *#lk* [, *SHFT*], *src* [, *dst*]	dst = src + #lk << SHFT	2	2	2	4-4
ADD *#lk*, 16, *src* [, *dst*]	dst = src + #lk << 16	2	2	2	4-4
ADD *src* [, *SHIFT*] [, *dst*]	dst = dst + src << SHIFT	1	1	1	4-4
ADD *src*, ASM [, *dst*]	dst = dst + src << ASM	1	1	1	4-4
ADDC *Smem*, *src*	src = src + Smem + C	1	1	3A, 3B	4-8
ADDM *#lk*, *Smem*	Smem = Smem + #lk	2	2	18A, 18B	4-9
ADDS *Smem*, *src*	src = src + uns(Smem)	1	1	3A, 3B	4-10

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–2. Subtract Instructions

Syntax	Expression	W†	Cycles†	Class	Page
SUB *Smem*, *src*	src = src – Smem	1	1	3A, 3B	4-187
SUB *Smem*, TS, *src*	src = src – Smem << TS	1	1	3A, 3B	4-187
SUB *Smem*, 16, *src* [, *dst*]	dst = src – Smem << 16	1	1	3A, 3B	4-187
SUB *Smem* [, *SHIFT*], *src* [, *dst*]	dst = src – Smem << SHIFT	2	2	4A, 4B	4-187
SUB *Xmem*, *SHFT*, *src*	src = src – Xmem << SHFT	1	1	3A	4-187
SUB *Xmem*, *Ymem*, *dst*	dst = Xmem << 16 – Ymem << 16	1	1	7	4-187
SUB #*lk* [, *SHFT*],*src* [, *dst*]	dst = src – #lk << SHFT	2	2	2	4-187
SUB #*lk*, 16, *src* [, *dst*]	dst = src – #lk <<16	2	2	2	4-187
SUB *src*[, *SHIFT*] [, *dst*]	dst = dst – src << SHIFT	1	1	1	4-187
SUB *src*, ASM [, *dst*]	dst = dst – src << ASM	1	1	1	4-187
SUBB *Smem*, *src*	src = src – Smem – $\overline{C}$	1	1	3A, 3B	4-191

SUBC *Smem*, *src*	If (src – Smem << 15) ≥ 0
	src = (src – Smem << 15) << 1 + 1
	Else
	src = src << 1
		1	1	3A, 3B	4-192

SUBS *Smem*, *src*	src = src – uns(Smem)	1	1	3A, 3B	4-194

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–3. Multiply Instructions

Syntax	Expression	W†	Cycles†	Class	Page
MPY *Smem*, *dst*	dst = T * Smem	1	1	3A, 3B	4-101
MPYR *Smem*, *dst*	dst = rnd(T * Smem)	1	1	3A, 3B	4-101
MPY *Xmem*, *Ymem*, *dst*	dst = Xmem * Ymem, T = Xmem	1	1	7	4-101
MPY *Smem*, #*lk*, *dst*	dst = Smem * #lk , T = Smem	2	2	6A, 6B	4-101
MPY #*lk*, *dst*	dst = T * #lk	2	2	2	4-101
MPYA *dst*	dst = T * A(32–16)	1	1	1	4-104
MPYA *Smem*	B = Smem * A(32–16), T = Smem	1	1	3A, 3B	4-104
MPYU *Smem*, *dst*	dst = uns(T) * uns(Smem)	1	1	3A, 3B	4-106
SQUR *Smem*, *dst*	dst = Smem * Smem, T = Smem	1	1	3A, 3B	4-161
SQUR A, *dst*	dst = A(32–16) * A(32–16)	1	1	1	4-161

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–4. Multiply-Accumulate and Multiply-Subtract Instructions

Syntax	Expression	W†	Cycles†	Class	Page
MAC *Smem*, *src*	src = src + T * Smem	1	1	3A, 3B	4-82
MAC *Xmem*, *Ymem*, *src* [, *dst*]	dst = src + Xmem * Ymem, T = Xmem	1	1	7	4-82
MAC #*lk*, *src* [, *dst*]	dst = src + T * #lk	2	2	2	4-82

MAC *Smem*, #*lk*, *src* [, *dst*]	dst = src + Smem * #lk,
	T = Smem
		2	2	6A, 6B	4-82

MACR *Smem*, *src*	src = rnd(src + T * Smem)	1	1	3A, 3B	4-82
MACR *Xmem*, *Ymem*, *src* [, *dst*]	dst = rnd(src + Xmem * Ymem), T = Xmem	1	1	7	4-82
MACA *Smem* [, *B*]	B = B + Smem * A(32–16), T = Smem	1	1	3A, 3B	4-85

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–4. Multiply-Accumulate and Multiply-Subtract Instructions (Continued)

Syntax	Expression	W†	Cycles†	Class	Page
MACA T, *src* [, *dst*]	dst = src + T * A(32–16)	1	1	1	4-85
MACAR *Smem* [, *B*]	B = rnd(B + Smem * A(32–16)), T = Smem	1	1	3A, 3B	4-85
MACAR T, *src* [, *dst*]	dst = rnd(src + T * A(32–16))	1	1	1	4-85
MACD *Smem*, *pmad*, *src*	src = src + Smem * pmad, T = Smem, (Smem + 1) = Smem	2	3	23A, 23B	4-87
MACP *Smem*, *pmad*, *src*	src = src + Smem * pmad, T = Smem	2	3	22A, 22B	4-89
MACSU *Xmem*, *Ymem*, *src*	src = src + uns(Xmem) * Ymem, T = Xmem	1	1	7	4-91
MAS *Smem*, *src*	src = src – T * Smem	1	1	3A, 3B	4-94
MASR *Smem*, *src*	src = rnd(src – T * Smem)	1	1	3A, 3B	4-94
MAS *Xmem*, *Ymem*, *src* [, *dst*]	dst = src – Xmem * Ymem, T = Xmem	1	1	7	4-94
MASR *Xmem*, *Ymem*, *src* [, *dst*]	dst = rnd(src – Xmem * Ymem), T = Xmem	1	1	7	4-94
MASA *Smem* [, *B*]	B = B – Smem * A(32–16), T = Smem	1	1	3A, 3B	4-97
MASA T, *src* [, *dst*]	dst = src – T * A(32–16)	1	1	1	4-97
MASAR T, *src* [, *dst*]	dst = rnd(src – T * A(32–16))	1	1	1	4-97
SQURA *Smem*, *src*	src = src + Smem * Smem, T = Smem	1	1	3A, 3B	4-163
SQURS *Smem, src*	src = src – Smem * Smem, T = Smem	1	1	3A, 3B	4-164

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–5. Double (32-Bit Operand) Instructions

Syntax	Expression	W†	Cycles†	Class	Page
DADD *Lmem*, *src* [, *dst*]	If C16 = 0 dst = Lmem + src If C16 = 1 dst(39–16) = Lmem(31–16) + src(31–16) dst(15–0) = Lmem(15–0) + src(15–0)	1	1	9A, 9B	4-37
DADST *Lmem*, *dst*	If C16 = 0 dst = Lmem + (T << 16 + T) If C16 = 1 dst(39–16) = Lmem(31–16) + T dst(15–0) = Lmem(15–0) – T	1	1	9A, 9B	4-39
DRSUB *Lmem*, *src*	If C16 = 0 src = Lmem – src If C16 = 1 src(39–16) = Lmem(31–16) – src(31–16) src(15–0) = Lmem(15–0) – src(15–0)	1	1	9A, 9B	4-43
DSADT *Lmem*, *dst*	If C16 = 0 dst = Lmem – (T << 16 + T) If C16 = 1 dst(39–16) = Lmem(31–16) – T dst(15–0) = Lmem(15–0) + T	1	1	9A, 9B	4-45
DSUB *Lmem*, *src*	If C16 = 0 src = src – Lmem If C16 = 1 src (39–16) = src(31–16) – Lmem(31–16) src (15–0) = src(15–0) – Lmem(15–0)	1	1	9A, 9B	4-48
DSUBT *Lmem*, *dst*	If C16 = 0 dst = Lmem – (T << 16 + T) If C16 = 1 dst(39–16) = Lmem(31–16) – T dst(15–0) = Lmem(15–0) – T	1	1	9A, 9B	4-50

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Lmem*.

Table 2–6. Application-Specific Instructions

Syntax	**Expression**	**W†**	**Cycles†**	**Class**	**Page**
ABDST *Xmem*, *Ymem*	B = B + |A(32–16)| A = (Xmem – Ymem) << 16	1	1	7	4-2
ABS *src* [, *dst*]	dst = |src|	1	1	1	4-3
CMPL *src* [, *dst*]	dst = ~src	1	1	1	4-32
DELAY *Smem*	(Smem + 1) = Smem	1	1	24A, 24B	4-41
EXP *src*	T = number of sign bits (src) – 8	1	1	1	4-52
FIRS *Xmem*, *Ymem*, *pmad*	B = B + A * pmad A = (Xmem + Ymem) << 16	2	3	8	4-59
LMS *Xmem*, *Ymem*	B = B + Xmem * Ymem A = A + Xmem << 16 + 2^{15}	1	1	7	4-80
MAX *dst*	dst = max(A, B)	1	1	1	4-99
MIN *dst*	dst = min(A, B)	1	1	1	4-100
NEG *src* [, *dst*]	dst = –src	1	1	1	4-119
NORM *src* [, *dst*]	dst = src << TS dst = norm(src, TS)	1	1	1	4-122
POLY *Smem*	B = Smem << 16 A = rnd(A(32–16) * T + B)	1	1	3A, 3B	4-126
RND *src* [, *dst*]	dst = src + 2^{15}	1	1	1	4-142
SAT *src*	saturate(src)	1	1	1	4-154
SQDST *Xmem*, *Ymem*	B = B + A(32–16) * A(32–16) A = (Xmem – Ymem) << 16	1	1	7	4-160

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

2.2 Logical Operations

This section summarizes the logical operation instructions. Table 2–7 through Table 2–11 list the instructions within the following functional groups:

- AND instructions (Table 2–7)
- OR instructions (Table 2–8 on page 2-8)
- XOR instructions (Table 2–9 on page 2-9)
- Shift instructions (Table 2–10 on page 2-9)
- Test instructions (Table 2–11 on page 2-9)

Table 2–7. AND Instructions

Syntax	Expression	W†	Cycles†	Class	Page
AND *Smem*, *src*	src = src & Smem	1	1	3A, 3B	4-11
AND #*lk* [, *SHFT*], *src* [, *dst*]	dst = src & #lk << SHFT	2	2	2	4-11
AND #*lk*, 16, *src* [, *dst*]	dst = src & #lk << 16	2	2	2	4-11
AND *src* [, *SHIFT*] [, *dst*]	dst = dst & src << SHIFT	1	1	1	4-11
ANDM #*lk*, *Smem*	Smem = Smem & #lk	2	2	18A, 18B	4-13

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–8. OR Instructions

Syntax	Expression	W†	Cycles†	Class	Page
OR *Smem, src*	src = src | Smem	1	1	3A, 3B	4-123
OR #*lk* [, *SHFT*], *src* [, *dst*]	dst = src | #lk << SHFT	2	2	2	4-123
OR #*lk,* 16, *src* [, *dst*]	dst = src | #lk << 16	2	2	2	4-123
OR *src* [, *SHIFT*] [, *dst*]	dst = dst | src << SHIFT	1	1	1	4-123
ORM #*lk, Smem*	Smem = Smem | #lk	2	2	18A, 18B	4-125

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–9. XOR Instructions

Syntax	Expression	W†	Cycles†	Class	Page
XOR *Smem, src*	src = src ^ Smem	1	1	3A, 3B	4-201
XOR *#lk* [, *SHFT,*], *src* [, *dst*]	dst = src ^ #lk << SHFT	2	2	2	4-201
XOR *#lk,* 16, *src* [, *dst*]	dst = src ^ #lk << 16	2	2	2	4-201
XOR *src* [, *SHIFT*] [, *dst*]	dst = dst ^ src << SHIFT	1	1	1	4-201
XORM *#lk, Smem*	Smem = Smem ^ #lk	2	2	18A, 18B	4-203

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–10. Shift Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ROL *src*	Rotate left with carry in	1	1	1	4-143
ROLTC *src*	Rotate left with TC in	1	1	1	4-144
ROR *src*	Rotate right with carry in	1	1	1	4-145
SFTA *src*, *SHIFT* [, *dst*]	dst = src << SHIFT {arithmetic shift}	1	1	1	4-155
SFTC *src*	if src(31) = src(30) then src = src << 1	1	1	1	4-157
SFTL *src*, *SHIFT* [, *dst*]	dst = src << SHIFT {logical shift}	1	1	1	4-158

† Values for words (W) and cycles assume the use of DARAM for data.

Table 2–11. Test Instructions

Syntax	Expression	W†	Cycles†	Class	Page
BIT *Xmem*, *BITC*	TC = Xmem(15 – BITC)	1	1	3A	4-21
BITF *Smem*, *#lk*	TC = (Smem && #lk)	2	2	6A, 6B	4-22
BITT *Smem*	TC = Smem(15 – T(3–0))	1	1	3A, 3B	4-23
CMPM *Smem, #lk*	TC = (Smem == #lk)	2	2	6A, 6B	4-33
CMPR *CC*, *ARx*	Compare ARx with AR0	1	1	1	4-34

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

2.3 Program-Control Operations

This section summarizes the program-control instructions. Table 2–12 through Table 2–18 list the instructions within the following functional groups:

- Branch instructions (Table 2–12)
- Call instructions (Table 2–13 on page 2-11)
- Interrupt instructions (Table 2–14 on page 2-11)
- Return instructions (Table 2–15 on page 2-12)
- Repeat instructions (Table 2–16 on page 2-12)
- Stack-manipulating instructions (Table 2–17 on page 2-13)
- Miscellaneous program-control instructions (Table 2–18 on page 2-13)

Table 2–12. Branch Instructions

Syntax	Expression	W†	Cycles†	Class	Page
B[D] *pmad*	PC = pmad(15–0)	2	4/[2¶]	29A	4-14
BACC[D] *src*	PC = src(15–0)	1	6/[4¶]	30A	4-15
BANZ[D] *pmad*, *Sind*	if (Sind ≠ 0) then PC = pmad(15–0)	2	4‡/2§/ [2¶]	29A	4-16
BC[D] *pmad*, *cond* [, *cond* [, *cond*]]	if (cond(s)) then PC = pmad(15–0)	2	5‡/3§/ [3¶]	31A	4-18
FB[D] *extpmad*	PC = pmad(15–0), XPC = pmad(22–16)	2	4/[2¶]	29A	4-53
FBACC[D] *src*	PC = src(15–0), XPC = src(22–16)	1	6/[4¶]	30A	4-54

† Values for words (W) and cycles assume the use of DARAM for data.
‡ Conditions true
§ Condition false
¶ Delayed instruction

Table 2–13. Call Instructions

Syntax	Expression	W†	Cycles†	Class	Page
CALA[D] *src*	– –SP, PC + 1[3¶] = TOS, PC = src(15–0)	1	6/[4¶]	30B	4-25
CALL[D] *pmad*	– –SP, PC + 2[4¶] = TOS, PC = pmad(15–0)	2	4/[2§]	29B	4-27
CC[D] *pmad*, *cond* [, *cond* [, *cond*]]	if (cond(s)) then – –SP, PC + 2[4¶] = TOS, PC = pmad(15–0)	2	5‡/3§/ [3¶]	31B	4-29
FCALA[D] *src*	– –SP, PC + 1 [3¶] = TOS, PC = src(15–0), XPC = src(22–16)	1	6/[4¶]	30B	4-55
FCALL[D] *extpmad*	– –SP, PC + 2[4¶] = TOS, PC = pmad(15–0), XPC = pmad(22–16)	2	4/[2¶]	29B	4-57

† Values for words (W) and cycles assume the use of DARAM for data.
‡ Conditions true
§ Condition false
¶ Delayed instruction

Table 2–14. Interrupt Instructions

Syntax	Expression	W†	Cycles†	Class	Page
INTR *K*	– –SP, + + PC = TOS, PC = IPTR(15–7) + K << 2, INTM = 1	1	3	35	4-65
TRAP *K*	– –SP, + + PC = TOS, PC = IPTR(15–7) + K << 2	1	3	35	4-195

† Values for words (W) and cycles assume the use of DARAM for data.

Table 2–15. Return Instructions

Syntax	Expression	W†	Cycles†	Class	Page
FRET[D]	XPC = TOS, ++ SP, PC = TOS, ++SP	1	6/[4¶]	34	4-61
FRETE[D]	XPC = TOS, ++ SP, PC = TOS, ++SP, INTM = 0	1	6/[4¶]	34	4-62
RC[D] *cond* [, *cond* [, *cond*]]	if (cond(s)) then PC = TOS, ++SP	1	5‡/3§/[3¶]	32	4-133
RET[D]	PC = TOS, ++SP	1	5/[3¶]	32	4-139
RETE[D]	PC = TOS, ++SP, INTM = 0	1	5/[3¶]	32	4-140
RETF[D]	PC = RTN, ++SP, INTM = 0	1	3/[1¶]	33	4-141

† Values for words (W) and cycles assume the use of DARAM for data.
‡ Conditions true
§ Condition false
¶ Delayed instruction

Table 2–16. Repeat Instructions

Syntax	Expression	W†	Cycles†	Class	Page
RPT *Smem*	Repeat single, RC = Smem	1	3	5A, 5B	4-146
RPT *#K*	Repeat single, RC = #K	1	1	1	4-146
RPT *#lk*	Repeat single, RC = #lk	2	2	2	4-146
RPTB[D] *pmad*	Repeat block, RSA = PC + 2[4¶], REA = pmad, BRAF = 1	2	4/[2¶]	29A	4-148
RPTZ *dst, #lk*	Repeat single, RC = #lk, dst = 0	2	2	2	4-150

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.
¶ Delayed instruction

Table 2–17. Stack-Manipulating Instructions

Syntax	Expression	W†	Cycles†	Class	Page
FRAME *K*	SP = SP + K	1	1	1	4-60
POPD *Smem*	Smem = TOS, ++SP	1	1	17A, 17B	4-127
POPM *MMR*	MMR = TOS, ++SP	1	1	17A	4-128
PSHD *Smem*	– –SP, Smem = TOS	1	1	16A, 16B	4-131
PSHM *MMR*	– –SP, MMR = TOS	1	1	16A	4-132

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–18. Miscellaneous Program-Control Instructions

Syntax	Expression	W†	Cycles†	Class	Page
IDLE *K*	idle(K)	1	4	36	4-63
MAR *Smem*	If CMPT = 0, then modify ARx If CMPT = 1 and ARx ≠ AR0, then modify ARx, ARP = x If CMPT = 1 and ARx = AR0, then modify AR(ARP)	1	1	1, 2	4-92
NOP	no operation	1	1	1	4-121
RESET	software reset	1	3	35	4-138
RSBX *N*, *SBIT*	STN (SBIT) = 0	1	1	1	4-151
SSBX *N*, *SBIT*	STN (SBIT) = 1	1	1	1	4-166
XC *n* , *cond* [, *cond*[, *cond*]]	If (cond(s)) then execute the next n instructions; n = 1 or 2	1	1	1	4-198

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

2.4 Load and Store Operations

This section summarizes the load and store instructions. Table 2–19 through Table 2–26 list the instructions within the following functional groups:

- Load instructions (Table 2–19)
- Store instructions (Table 2–20 on page 2-15)
- Conditional store instructions (Table 2–21 on page 2-16)
- Parallel load and store instructions (Table 2–22 on page 2-16)
- Parallel load and multiply instructions (Table 2–23 on page 2-16)
- Parallel store and add/subtract instructions (Table 2–24 on page 2-17)
- Parallel store and multiply instructions (Table 2–25 on page 2-17)
- Miscellaneous load-type and store-type instructions (Table 2–26 on page 2-18)

Table 2–19. Load Instructions

Syntax	Expression	W†	Cycles†	Class	Page
DLD *Lmem*, *dst*	dst = Lmem	1	1	9A, 9B	4-42
LD *Smem*, *dst*	dst = Smem	1	1	3A, 3B	4-66
LD *Smem*, TS, *dst*	dst = Smem << TS	1	1	3A, 3B	4-66
LD *Smem,* 16*, dst*	dst = Smem << 16	1	1	3A, 3B	4-66
LD *Smem* [, *SHIFT*], *dst*	dst = Smem << SHIFT	2	2	4A, 4B	4-66
LD *Xmem*, *SHFT*, *dst*	dst = Xmem << SHFT	1	1	3A	4-66
LD #*K, dst*	dst = #K	1	1	1	4-66
LD #*lk* [, *SHFT*], *dst*	dst = #lk << SHFT	2	2	2	4-66
LD #*lk*, 16, *dst*	dst = #lk << 16	2	2	2	4-66
LD *src*, ASM [, *dst*]	dst = src << ASM	1	1	1	4-66
LD *src* [, *SHIFT*], *dst*	dst = src << SHIFT	1	1	1	4-66
LD *Smem,* T	T = Smem	1	1	3A, 3B	4-70
LD *Smem,* DP	DP = Smem(8–0)	1	3	5A, 5B	4-70
LD #*k9*, DP	DP = #k9	1	1	1	4-70
LD #*k5*, ASM	ASM = #k5	1	1	1	4-70

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Lmem* or *Smem*.

Table 2–19. Load Instructions (Continued)

Syntax	Expression	W†	Cycles†	Class	Page
LD *#k3*, ARP	ARP = #k3	1	1	1	4-70
LD *Smem,* ASM	ASM = Smem(4–0)	1	1	3A, 3B	4-70
LDM *MMR, dst*	dst = MMR	1	1	3A	4-73
LDR *Smem*, *dst*	dst = rnd(Smem)	1	1	3A, 3B	4-78
LDU *Smem*, *dst*	dst = uns(Smem)	1	1	3A, 3B	4-79
LTD *Smem*	T = Smem, (Smem + 1) = Smem	1	1	24A, 24B	4-81

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Lmem* or *Smem*.

Table 2–20. Store Instructions

Syntax	Expression	W†	Cycles†	Class	Page
DST *src*, *Lmem*	Lmem = src	1	2	13A, 13B	4-47
ST T, *Smem*	Smem = T	1	1	10A, 10B	4-167
ST TRN, *Smem*	Smem = TRN	1	1	10A, 10B	4-167
ST *#lk*, *Smem*	Smem = #lk	2	2	12A, 12B	4-167
STH *src*, *Smem*	Smem = src << –16	1	1	10A, 10B	4-169
STH *src*, ASM, *Smem*	Smem = src << (ASM – 16)	1	1	10A, 10B	4-169
STH *src*, *SHFT*, *Xmem*	Xmem = src << (SHFT – 16)	1	1	10A	4-169
STH *src* [, *SHIFT*], *Smem*	Smem = src << (SHIFT – 16)	2	2	11A, 11B	4-169
STL *src*, *Smem*	Smem = src	1	1	10A, 10B	4-172
STL *src*, ASM, *Smem*	Smem = src << ASM	1	1	10A, 10B	4-172
STL *src*, *SHFT*, *Xmem*	Xmem = src << SHFT	1	1	10A, 10B	4-172
STL *src* [, *SHIFT*], *Smem*	Smem = src << SHIFT	2	2	11A, 11B	4-172
STLM *src*, *MMR*	MMR = src	1	1	10A	4-175
STM *#lk*, *MMR*	MMR = #lk	2	2	12A	4-176

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Lmem* or *Smem*.

Table 2–21. Conditional Store Instructions

Syntax	Expression	W†	Cycles†	Class	Page

CMPS *src, Smem*	If src(31–16) > src(15–0) then
	Smem = src(31–16)
	If src(31–16) ≤ src(15–0) then
	Smem = src(15–0)
		1	1	10A, 10B	4-35

SACCD *src, Xmem, cond*	If (cond) Xmem = src << (ASM – 16)	1	1	15	4-152
SRCCD *Xmem*, *cond*	If (cond) Xmem = BRC	1	1	15	4-165
STRCD *Xmem*, *cond*	If (cond) Xmem = T	1	1	15	4-186

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–22. Parallel Load and Store Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ST *src*, *Ymem* || LD *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = Xmem << 16	1	1	14	4-178
ST *src*, *Ymem* || LD *Xmem*, T	Ymem = src << (ASM – 16) || T = Xmem	1	1	14	4-178

† Values for words (W) and cycles assume the use of DARAM for data.

Table 2–23. Parallel Load and Multiply Instructions

Syntax	Expression	W†	Cycles†	Class	Page
LD *Xmem*, *dst* || MAC *Ymem*, *dst_*	dst = Xmem << 16 || dst_ = dst_ + T * Ymem	1	1	7	4-74
LD *Xmem*, *dst* || MACR *Ymem*, *dst_*	dst = Xmem << 16 || dst_ = rnd(dst_ + T * Ymem)	1	1	7	4-74
LD *Xmem*, *dst* || MAS *Ymem*, *dst_*	dst = Xmem << 16 || dst_ = dst_ – T * Ymem	1	1	7	4-76
LD *Xmem*, *dst* || MASR *Ymem*, *dst_*	dst = Xmem << 16 || dst_ = rnd(dst_ – T * Ymem)	1	1	7	4-76

† Values for words (W) and cycles assume the use of DARAM for data.

Table 2–24. Parallel Store and Add/Subtract Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ST *src*, *Ymem* || ADD *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = dst_ + Xmem << 16	1	1	14	4-177
ST *src*, *Ymem* || SUB *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = (Xmem << 16) – dst_	1	1	14	4-185

† Values for words (W) and cycles assume the use of DARAM for data.

Table 2–25. Parallel Store and Multiply Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ST *src*, *Ymem* || MAC *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = dst + T * Xmem	1	1	14	4-180
ST *src*, *Ymem* || MACR *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = rnd(dst + T * Xmem)	1	1	14	4-180
ST *src*, *Ymem* || MAS *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = dst – T * Xmem	1	1	14	4-182
ST *src*, *Ymem* || MASR *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = rnd(dst – T * Xmem)	1	1	14	4-182
ST *src*, *Ymem* || MPY *Xmem*, *dst*	Ymem = src << (ASM – 16) || dst = T * Xmem	1	1	14	4-184

† Values for words (W) and cycles assume the use of DARAM for data.

Table 2–26. Miscellaneous Load-Type and Store-Type Instructions

Syntax	Expression	W†	Cycles†	Class	Page
MVDD *Xmem*, *Ymem*	Ymem = Xmem	1	1	14	4-107
MVDK *Smem*, *dmad*	dmad = Smem	2	2	19A, 19B	4-108
MVDM *dmad*, *MMR*	MMR = dmad	2	2	19A	4-110
MVDP *Smem*, *pmad*	pmad = Smem	2	4	20A, 20B	4-111
MVKD *dmad*, *Smem*	Smem = dmad	2	2	19A, 19B	4-113
MVMD *MMR*, *dmad*	dmad = MMR	2	2	19A	4-115
MVMM *MMRx*, *MMRy*	MMRy = MMRx	1	1	1	4-116
MVPD *pmad*, *Smem*	Smem = pmad	2	3	21A, 21B	4-117
PORTR *PA*, *Smem*	Smem = PA	2	2	27A, 27B	4-129
PORTW *Smem, PA*	PA = Smem	2	2	28A, 28B	4-130
READA *Smem*	Smem = A	1	5	25A, 25B	4-136
WRITA *Smem*	A = Smem	1	5	26A, 26B	4-196

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

2.5 Repeating a Single Instruction

The TMS320C54x™ DSP includes repeat instructions that cause the next instruction to be repeated. The number of times for the instruction to be repeated is obtained from an operand of the instruction and is equal to this operand + 1. This value is stored in the 16-bit repeat counter (RC) register. You cannot program the value in the RC register; it is loaded by the repeat instructions only. The maximum number of executions of a given instruction is 65 536. An absolute program or data address is automatically incremented when the single-repeat feature is used.

Once a repeat instruction is decoded, all interrupts, including $\overline{\text{NMI}}$ but not $\overline{\text{RS}}$, are disabled until the completion of the repeat loop. However, the C54x™ DSP does respond to the $\overline{\text{HOLD}}$ signal while executing a repeat loop—the response depends on the value of the HM bit of status register 1 (ST1).

The repeat function can be used with some instructions, such as multiply/accumulate and block moves, to increase the execution speed of these instructions. These multicycle instructions (Table 2–27) effectively become single-cycle instructions after the first iteration of a repeat instruction.

Table 2–27. Multicycle Instructions That Become Single-Cycle Instructions When Repeated

Instruction	Description	# Cycles†
FIRS	Symmetrical FIR filter	3
MACD	Multiply and move result in accumulator with delay	3
MACP	Multiply and move result in accumulator	3
MVDK	Data-to-data move	2
MVDM	Data-to-MMR move	2
MVDP	Data-to-program move	4
MVKD	Data-to-data move	2
MVMD	MMR-to-data move	2
MVPD	Program-to-data move	3
READA	Read from program-memory to data memory	5
WRITA	Write data memory to program memory	5

† Number of cycles when instruction is not repeated

Single data-memory operand instructions cannot be repeated if a long offset modifier or an absolute address is used (for example, *ARn(lk), *+ARn(lk), *+ARn(lk)% and *(lk)). Instructions listed in Table 2–28 cannot be repeated using RPT or RPTZ instructions.

Table 2–28. Nonrepeatable Instructions

Instruction	Description
ADDM	Add long constant to data memory
ANDM	AND data memory with long constant
B[D]	Unconditional branch
BACC[D]	Branch to accumulator address
BANZ[D]	Branch on auxiliary register not 0
BC[D]	Conditional branch
CALA[D]	Call to accumulator address
CALL[D]	Unconditional call
CC[D]	Conditional call
CMPR	Compare with auxiliary register
DST	Long word (32-bit) store
FB[D]	Far branch unconditionally
FBACC[D]	Far branch to location specified by accumulator
FCALA[D]	Far call subroutine at location specified by accumulator
FCALL[D]	Far call unconditionally
FRET[D]	Far return
FRETE[D]	Enable interrupts and far return from interrupt
IDLE	Idle instructions
INTR	Interrupt trap
LD ARP	Load auxiliary register pointer (ARP)
LD DP	Load data page pointer (DP)
MVMM	Move memory-mapped register (MMR) to another MMR
ORM	OR data memory with long constant

Table 2–28. Nonrepeatable Instructions (Continued)

Instruction	Description
RC[D]	Conditional return
RESET	Software reset
RET[D]	Unconditional return
RETE[D]	Return from interrupt
RETF[D]	Fast return from interrupt
RND	Round accumulator
RPT	Repeat next instruction
RPTB[D]	Block repeat
RPTZ	Repeat next instruction and clear accumulator
RSBX	Reset status register bit
SSBX	Set status register bit
TRAP	Software trap
XC	Conditional execute
XORM	XOR data memory with long constant

Chapter 3

Instruction Classes and Cycles

Instructions are classified into several categories, or classes, according to cycles required. This chapter describes the instruction classes. Because a single instruction can have multiple syntaxes and types of execution, it can appear in multiple classes.

The tables in this chapter show the number of cycles required for a given TMS320C54x™ DSP instruction to execute in a given memory configuration when executed as a single instruction and when executed in the repeat mode. Tables are also provided for a single data-memory operand access used with a long constant. The column headings in the tables indicate the program source location. These headings are defined as follows:

ROM The instruction executes from internal program ROM.

SARAM The instruction executes from internal single-access RAM.

DARAM The instruction executes from internal dual-access RAM.

External The instruction executes from external program memory.

If a class of instructions requires memory operand(s), the row divisions in the tables indicate the location(s) of the operand(s). These locations are defined as follows:

DARAM The operand is in internal dual-access RAM.

SARAM The operand is in internal single-access RAM.

DROM The operand is in internal data ROM.

PROM The operand is in internal program ROM.

External The operand is in external memory.

MMR The operand is a memory-mapped register.

The number of cycles required for each instruction is given in terms of the processor machine cycles (the CLKOUT period). The additional wait states for program/data memory accesses and I/O accesses are defined as follows:

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

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

n Repetitions—the number of times a repeated instruction is executed.

nd Data-memory wait states repeated n times.

np Program-memory wait states repeated n times.

npd Program-memory wait states repeated n times.

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

pd Program-memory wait states—the number of additional clock cycles the device waits for external program memory to respond to an access as a program data operand.

These variables can also use the subscripts src, dst, and code to indicate source, destination, and code, respectively.

All reads from external memory take at least one instruction cycle to complete, and all writes to external memory take at least two instruction cycles to complete. These external accesses take longer if additional wait-state cycles are added using the software wait-state generator or the external READY input. However, internal to the CPU all writes to external memory take only one cycle as long as no other access to the external memory is in process at the same time. This is possible because the instruction pipeline takes only one cycle to request an external write access, and the external bus interface unit completes the write access independently.

The instruction cycles are based on the following assumptions:

- ❑ At least five instructions following the current instruction are fetched from the same memory section (internal or external) as the current instruction, except in instructions that cause a program counter (PC) discontinuity, such as a branch or call.
- ❑ When executing a single instruction, there is no pipeline or bus conflict between the current instruction and any other instruction in the pipeline. The only exception is the conflict between the instruction fetch and the memory read/write access (if any) of the instruction under consideration.
- ❑ In single-instruction repeat mode, all conflicts caused by the pipelined execution of that instruction are considered.

Class 1

1 word, 1 cycle. No operand, or short-immediate or register operands and no memory operands.

Mnemonics

ABS	MACA[R]	NORM	SFTA
ADD	MAR	OR	SFTC
AND	MASA[R]	RND	SFTL
CMPL	MAX	ROL	SQUR
CMPR	MIN	ROLTC	SSBX
EXP	MPYA	ROR	SUB
FRAME	MVMM	RPT	XC
LD	NEG	RSBX	XOR
LD T/DP/ASM/ARP	NOP	SAT	

Cycles

Cycles for a Single Execution

Program		
ROM/SARAM	**DARAM**	**External**
1	1	1+p

Cycles for a Repeat Execution

Program		
ROM/SARAM	**DARAM**	**External**
n	n	n+p

Class 2

2 words, 2 cycles. Long-immediate operand and no memory operands.

Mnemonics

ADD	MAC	OR	SUB
AND	MAR	RPT	XOR
LD	MPY	RPTZ	

Cycles

Cycles for a Single Execution

Program		
ROM/SARAM	**DARAM**	**External**
2	2	2+2p

Cycles for a Repeat Execution

Program		
ROM/SARAM	**DARAM**	**External**
n+1	n+1	n+1+2p

Class 3A

1 word, 1 cycle. Single data-memory (Smem or Xmem) read operand or MMR read operand.

Mnemonics

ADD	LDM	MPYA	SUBB
ADDC	LDR	MPYU	SUBC
ADDS	LDU	OR	SUBS
AND	MAC[R]	POLY	XOR
BIT	MACA[R]	SQUR	
BITT	MAS[R]	SQURA	
LD	MASA	SQURS	
LD T/DP/ASM/ARP	MPY[R]	SUB	

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	1	1, 2†	1+p
SARAM	1, 2†	1	1+p
DROM	1, 2†	1	1+p
External	1+d	1+d	2+d+p
MMR◊	1	1	1+p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	n	n, n+1†	n+p
SARAM	n, n+1†	n	n+p
DROM	n, n+1†	n	n+p
External	n+nd	n+nd	n+1+nd+p
MMR◊	n	n	n+p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 3B

2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing.

Mnemonics

ADD	LDU	OR	SUBS
ADDC	MAC[R]	POLY	XOR
ADDS	MACA[R]	SQUR	
AND	MAS[R]	SQURA	
BITT	MASA	SQURS	
LD	MPY[R]	SUB	
LD T/DP/ASM/ARP	MPYA	SUBB	
LDR	MPYU	SUBC	

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2, 3†	2+2p
SARAM	2, 3†	2	2+2p
DROM	2, 3†	2	2+2p
External	2+d	2+d	3+d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 4A

2 words, 2 cycles. Single data-memory (Smem) read operand.

Mnemonics

ADD LD SUB

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2, 3†	2+2p
SARAM	2, 3†	2	2+2p
DROM	2, 3†	2	2+2p
External	2+d	2+d	3+d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	n+1	n+1, n+2†	n+1+2p
SARAM	n+1, n+2†	n+1	n+1+2p
DROM	n+1, n+2†	n+1	n+1+2p
External	n+1+nd	n+1+nd	n+2+nd+2p
MMR◊	n+1	n+1	n+1+2p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 4B

3 words, 3 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing.

Mnemonics

ADD LD SUB

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3, 4†	3+3p
SARAM	3, 4†	3	3+3p
DROM	3, 4†	3	3+3p
External	3+d	3+d	4+d+3p
MMR◊	3	3	3+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 5A

1 word, 3 cycles. Single data-memory (Smem) read operand (with DP destination for load instruction).

Mnemonics

LD RPT

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3	3+p
SARAM	3	3	3+p
DROM	3	3	3+p
External	3+d	3+d	3+d+p
MMR◊	3	3	3+p

◊ Add one cycle for peripheral memory-mapped access.

Class 5B

2 words, 4 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing (with DP destination for load instruction).

Mnemonics

LD RPT

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	4	4	4+2p
SARAM	4	4	4+2p
DROM	4	4	4+2p
External	4+d	4+d	4+d+2p
MMR◊	4	4	4+2p

◊ Add one cycle for peripheral memory-mapped access.

Class 6A

2 words, 2 cycles. Single data-memory (Smem) read operand and single long-immediate operand.

Mnemonics

BITF CMPM MAC MPY

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2, 3†	2+2p
SARAM	2, 3†	2	2+2p
DROM	2, 3†	2	2+2p
External	2+d	2+d	3+d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	n+1	n+1, n+2†	n+1+2p
SARAM	n+1, n+2†	n+1	n+1+2p
DROM	n+1, n+2†	n+1	n+1+2p
External	n+1+nd	n+1+nd	n+2+nd+2p
MMR◊	n+1	n+1	n+1+2p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 6B

3 words, 3 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single long-immediate operand.

Mnemonics

BITF CMPM MAC MPY

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	**Program**		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3, 4†	3+3p
SARAM	3, 4†	3	3+3p
DROM	3, 4†	3	3+3p
External	3+d	3+d	4+d+3p
MMR◊	3	3	3+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 7

1 word, 1 cycle. Dual data-memory (Xmem and Ymem) read operands.

Mnemonics

ABDST	LD||MAS[R]	MACSU	SQDST
ADD	LMS	MAS[R]	SUB
LD||MAC[R]	MAC[R]	MPY	

Cycles

Cycles for a Single Execution

Operand		**Program**		
Xmem	**Ymem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1, 2†	1+p
	DROM	1, 2†	1, 2†	1+p
	External	1+d	1+d, 2‖	2+d+p
SARAM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†, 3‡	1, 2†	1+p, 2☆
	DROM	1, 2†	1	1+p
	External	1+d, 2‖	1+d	2+d+p
DROM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†	1, 2†	1+p, 2☆
	DROM	1, 2†, 3‡	1, 2†	1+p, 2☆
	External	1+d, 2‖	1+d	2+d+p
External	DARAM	1+d	1+d	2+d+p
	SARAM	1+d, 2‖	1+d	2+d+p
	DROM	1+d, 2‖	1+d	2+d+p
	External	2+2d	2+2d	3+2d+p
MMR◊	DARAM	1	1	1+p
	SARAM	1, 2†	1	1+p
	DROM	1, 2†	1	1+p
	External	1+d	1+d	2+d+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
‖ One operand and code in same memory block when d = 0
☆ Two operands in same memory block when p = 0
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Xmem	**Ymem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n	n, n+1†	n+p
	SARAM	n, n+1†	n, n+1†	n+p
	DROM	n, n+1†	n, n+1†	n+p
	External	n+nd	n+nd, 1+n‖	n+1+nd+p
SARAM	DARAM	n, n+1†	n	n+p
	SARAM	n, n+1†, 2n#, 2n+1‡	n, 2n#	n+p, 2n (p = 0)#, 2n–1+p (p ≥ 1)#
	DROM	n, n+1†	n	n+p
	External	n+nd, n+1‖	n+nd	n+1+nd+p
DROM	DARAM	n, n+1†	n	n+p
	SARAM	n, n+1†	n	n+p
	DROM	n, n+1†, 2n#, 2n+1‡	n, 2n#	n+p, 2n (p = 0)#, 2n–1+p (p ≥ 1)#
	External	n+nd, n+1‖	n+nd	n+1+nd+p
External	DARAM	n+nd	n+nd	n+1+nd+p
	SARAM	n+nd, n+1‖	n+nd	n+1+nd+p
	DROM	n+nd, n+1‖	n+nd	n+1+nd+p
	External	2n+2nd	2n+2nd	2n+1+2nd+p
MMR◊	DARAM	n	n	n+p
	SARAM	n, n+1†	n	n+p
	DROM	n, n+1†	n	n+p
	External	n+nd	n+nd	n+1+nd+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
# Two operands in same memory block
‖ One operand and code in same memory block when d = 0
◊ Add n cycles for peripheral memory-mapped access.

Class 8

2 words, 3 cycles. Dual data-memory (Xmem and Ymem) read operands and a single program-memory (pmad) operand.

Mnemonics

FIRS

Cycles

Cycles for a Single Execution

Operand			Program		
pmad	**Xmem**	**Ymem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	DARAM	3, 4†	3, 4†	3+2p, 4+2p†
		SARAM/ DROM	3, 4†	3, 4†	3+2p, 4+2p†
		External	3+d, 4+d†	3+d, 4+d†	3+d+2p, 4+d+2p†
	SARAM/ DROM	DARAM	3	3	3+2p
		SARAM/ DROM	3, 4‡	3, 4‡	3+2p, 4+2p‡
		External	3+d	3+d	3+d+2p
	External	DARAM	3+d	3+d	3+d+2p
		SARAM/ DROM	3+d	3+d	3+d+2p
		External	4+2d	4+2d	4+2d+2p
SARAM/ DROM	DARAM	DARAM	3	3	3+2p
		SARAM/ DROM	3, 4§	3, 4§	3+2p, 4+2p§
		External	3+d	3+d	3+d+2p

† Xmem and pmad in same memory block
‡ Xmem and Ymem in same memory block
§ Ymem and pmad in same memory block
¶ Xmem, Ymem, and pmad in same memory block

Cycles for a Single Execution (Continued)

Operand			Program		
pmad	**Xmem**	**Ymem**	**ROM/SARAM**	**DARAM**	**External**
	SARAM/ DROM	DARAM	3, 4†	3, 4†	3+2p, 4+2p†
		SARAM/ DROM	3, 4†, 5¶	3, 4†, 5¶	3+2p, 4+2p†, 5+2p¶
		External	3+d, 4+d†	3+d, 4+d†	3+d+2p, 4+d+2p†
	External	DARAM	3+d	3+d	3+2p
		SARAM/ DROM	3+d, 4+d§	3+d, 4+d§	3+2p, 4+d+2p§
		External	4+2d	4+2d	4+2d+2p
External	DARAM	DARAM	3+pd	3+pd	3+pd+2p
		SARAM/ DROM	3+pd	3+pd	3+pd+2p
		External	4+pd+d	4+pd+d	4+pd+d+2p
	SARAM/ DROM	DARAM	3+pd	3+pd	3+pd+2p
		SARAM/ DROM	3+pd, 4+pd‡	3+pd, 4+pd‡	3+pd+2p, 4+pd+2p‡
		External	4+pd+d	4+pd+d	4+pd+d+2p
	External	DARAM	4+pd+d	4+pd+d	4+pd+d+2p
		SARAM/ DROM	4+pd+d	4+pd+d	4+pd+d+2p
		External	5+pd+2d	5+pd+2d	5+pd+2d +2p

† Xmem and pmad in same memory block
‡ Xmem and Ymem in same memory block
§ Ymem and pmad in same memory block
¶ Xmem, Ymem, and pmad in same memory block

Cycles for a Repeat Execution

Operand			Program		
pmad	**Xmem**	**Ymem**	**ROM/ SARAM**	**DARAM**	**External**
DARAM	DARAM	DARAM	n+2, 2n+2†	n+2, 2n+2†	n+2+2p, 2n+2+2p†
		SARAM/ DROM	n+2, 2n+2†	n+2, 2n+2†	n+2+2p, 2n+2+2p†
		External	n+2+nd, 2n+2+nd†	n+2+nd, 2n+2+nd†	n+2+nd+2p, 2n+2+nd +2p†
	SARAM/ DROM	DARAM	n+2	n+2	n+2+2p
		SARAM/ DROM	n+2, 2n+2‡	n+2, 2n+2‡	n+2+2p, 2n+2+2p‡
		External	n+2+nd	n+2+nd	n+2+nd+2p
	External	DARAM	n+2+nd	n+2+nd	n+2+nd+2p
		SARAM/ DROM	n+2+nd	n+2+nd	n+2+nd+2p
		External	2n+2+2nd	2n+2+2nd	2n+2+2nd +2p
SARAM/ DROM	DARAM	DARAM	n+2	n+2	n+2+2p
		SARAM/ DROM	n+2, 2n+2§	n+2, 2n+2§	n+2+2p, 2n+2+2p§
		External	n+2+nd	n+2+nd	n+2+nd+2p

† Xmem and pmad in same memory block
‡ Xmem and Ymem in same memory block
§ Ymem and pmad in same memory block
¶ Xmem, Ymem, and pmad in same memory block

Cycles for a Repeat Execution (Continued)

Operand			Program		
pmad	**Xmem**	**Ymem**	**ROM/ SARAM**	**DARAM**	**External**
	SARAM/ DROM	DARAM	n+2, 2n+2†	n+2, 2n+2†	n+2+2p, 2n+2+2p†
		SARAM/ DROM	n+2, 2n+2†, 3n+2¶	n+2, 2n+2†, 3n+2¶	n+2+2p, 2n+2+2p†, 3n+2+2p¶
		External	n+2+nd, 2n+2+nd†	n+2+nd, 2n+2+nd†	n+2+nd+2p, 2n+2+nd +2p†
	External	DARAM	n+2+nd	n+2+nd	n+2+nd
		SARAM/ DROM	n+2+nd, 2n+2+nd§	n+2+nd, 2n+2+nd§	n+2+nd+2p, 2n+2+nd +2p§
		External	2n+2+2nd	2n+2+2nd	2n+2+2nd +2p
External	DARAM	DARAM	n+2+npd	n+2+npd	n+2+npd+2p
		SARAM/ DROM	n+2+npd	n+2+npd	n+2+npd+2p
		External	2n+2+npd+nd	2n+2+npd+nd	2n+2+npd +nd+2p
	SARAM/ DROM	DARAM	n+2+npd	n+2+npd	n+2+npd+2p
		SARAM/ DROM	n+2+npd, 2n+2+npd‡	n+2+npd, 2n+2+npd‡	n+2+npd+2p, 2n+2+npd +2p‡
		External	2n+2+npd+nd	2n+2+npd+nd	2n+2+npd +nd+2p

† Xmem and pmad in same memory block
‡ Xmem and Ymem in same memory block
§ Ymem and pmad in same memory block
¶ Xmem, Ymem, and pmad in same memory block

Cycles for a Repeat Execution (Continued)

Operand			Program		
pmad	**Xmem**	**Ymem**	**ROM/ SARAM**	**DARAM**	**External**
	External	DARAM	2n+2+npd+nd	2n+2+npd+nd	2n+2+npd +nd+2p
		SARAM/ DROM	2n+2+npd+nd	2n+2+npd+nd	2n+2+npd +nd+2p
		External	3n+2+npd+2nd	3n+2+npd+2nd	3n+2+npd +2nd+2p

† Xmem and pmad in same memory block
‡ Xmem and Ymem in same memory block
§ Ymem and pmad in same memory block
¶ Xmem, Ymem, and pmad in same memory block

Class 9A

1 word, 1 cycle. Single long-word data-memory (Lmem) read operand.

Mnemonics

DADD	DLD	DSADT	DSUBT
DADST	DRSUB	DSUB	

Cycles

Cycles for a Single Execution

Operand	Program		
Lmem	**ROM/SARAM**	**DARAM**	**External**
DARAM	1	1, 2†	1+p
SARAM	1, 2†	1	1+p
DROM	1, 2†	1	1+p
External	2+2d	2+2d	3+2d+p

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand	Program		
Lmem	**ROM/SARAM**	**DARAM**	**External**
DARAM	n	n, n+1†	n+p
SARAM	n, n+1†	n	n+p
DROM	n, n+1†	n	n+p
External	2n+2nd	2n+2nd	1+2n+2nd+p

† Operand and code in same memory block

Class 9B

2 words, 2 cycles. Single long-word data-memory (Lmem) read operand using long-offset indirect addressing.

Mnemonics

DADD	DLD	DSADT	DSUBT
DADST	DRSUB	DSUB	

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Lmem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2, 3†	2+2p
SARAM	2, 3†	2	2+2p
DROM	2, 3†	2	2+2p
External	3+2d	3+2d	4+2d+2p

† Operand and code in same memory block

Class 10A

1 word, 1 cycle. Single data-memory (Smem or Xmem) write operand or an MMR write operand.

Mnemonics

CMPS	STH	STLM
ST	STL	

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	1	1	1+p
SARAM	1, 2†	1	1+p
External	1	1	4+d+p
MMR◊	1	1	1+p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	n	n	n+p
SARAM	n, n+1†	n	n+p
External	2n–1+(n–1)d	2n–1+(n–1)d	2n+2+nd+p
MMR◊	n	n	n+p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 10B

2 words, 2 cycles. Single data-memory (Smem or Xmem) write operand using long-offset indirect addressing.

Mnemonics

CMPS ST STH STL

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2	2+2p
SARAM	2, 3†	2	2+2p
External	2	2	5+d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 11A

2 words, 2 cycles. Single data-memory (Smem) write operand.

Mnemonics

STH   STL

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	2	2	2+2p
SARAM	2, 3†	2	2+2p
External	2	2	5+d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	n+1	n+1	n+1+2p
SARAM	n+1, n+2†	n+1	n+1+2p
External	2n+(n–1)d	2n+(n–1)d	2n+3+nd+2p
MMR◊	n+1	n+1	n+1+2p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 11B

3 words, 3 cycles. Single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics

STH STL

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3	3+3p
SARAM	3, 4†	3	3+3p
External	3	3	6+d+3p
MMR◊	3	3	3+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 12A

2 words, 2 cycles. Single data-memory (Smem) write operand or MMR write operand.

Mnemonics

ST STM

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2	2+2p
SARAM	2, 3†	2	2+2p
External	2	2	5+d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2n	2n	2n+2p
SARAM	2n, 2n+1†	2n	2n+2p
External	2n+(n–1)d	2n+(n–1)d	2n+3+nd+p
MMR◊	2n	2n	2n+2p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 12B

3 words, 3 cycles. Single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics

ST

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3	3+3p
SARAM	3, 4†	3	3+3p
External	3	3	6+d+3p
MMR◊	3	3	3+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 13A

1 word, 2 cycles. Single long-word data-memory (Lmem) write operand.

Mnemonics

DST

Cycles

Cycles for a Single Execution

Operand	Program		
Lmem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2	2+p
SARAM	2, 4†	2	2+p
External	3+d	3+d	8+2d+p
MMR◊	2	2	2+p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Lmem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2n	2n	2n+p
SARAM	2n, 2n+2†	2n	2n+p
External	4n–1+(2n–1)d	4n–1+(2n–1)d	4n+4+2nd+p
MMR◊	2n	2n	2n+p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 13B

2 words, 3 cycles. Single long-word data-memory (Lmem) write operand using long-offset indirect addressing.

Mnemonics

DST

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Lmem	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3	3+2p
SARAM	3, 5†	3	3+2p
External	4+d	4+d	9+2d+2p
MMR◊	3	3	3+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 14

1 word, 1 cycle. Dual data-memory (Xmem and Ymem) read and write operands.

Mnemonics

MVDD	ST‖LD	ST‖MAS[R]	ST‖SUB
ST‖ADD	ST‖MAC[R]	ST‖MPY	

Cycles

Cycles for a Single Execution

Operand		Program		
Xmem	**Ymem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1, 2†	1+p
	External	1	1, 2†	4+d+p
SARAM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†, 3‡	1	1+p
	External	1, 2†	1	4+d+p
DROM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†	1	1+p
	External	1, 2†	1	4+d+p
External	DARAM	1+d	1+d	2+d+p
	SARAM	1+d, 2+d†	1+d	2+d+p
	External	1+d	1+d	5+2d+p
MMR◊	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1	1+p
	External	1	1	4+d+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Xmem	**Ymem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n	n, n+1†	n+p
	SARAM	n, n+1†	n, n+1†	n+p
	External	2n–1+(n–1)d	2n–1+(n–1)d, 2n+(n–1)d†	2n+2+nd+p
SARAM	DARAM	n, n+1†	n	n+p
	SARAM	n, n+1†, 2n#, 2n+1‡	n, 2n#	n+p, 2n+p#
	External	2n–1+(n–1)d, 2n+(n–1)d†	2n–1+(n–1)d, 2n+(n–1)d†	2n+2+nd+p
DROM	DARAM	n, n+1†	n, n+1†	n+p
	SARAM	n, n+1†	n	n+p
	External	2n–1+(n–1)d, 2n+(n–1)d†	2n–1+(n–1)d	2n+2+nd+p
External	DARAM	n+nd	n+nd	n+1+nd+p
	SARAM	n+nd, n+1+nd†	n+nd	n+1+nd+p
	External	4n–3+(2n–1)d	4n–3+(2n–1)d	4n+1+2nd+p
MMR◊	DARAM	n	n, 2n†	n+p
	SARAM	n, n+1†	n	n+p
	External	2n–1+(n–1)d	2n–1+(n–1)d	2n+2+nd+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
# Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 15

1 word, 1 cycle. Single data-memory (Xmem) write operand.

Mnemonics

SACCD SRCCD STRCD

Cycles

Cycles for a Single Execution

Operand	Program		
Xmem	ROM/SARAM	DARAM	External
DARAM	1	1	1+p
SARAM	1, 2†	1	1+p
External	1	1	4+d+p
MMR◊	1	1	1+p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
Xmem	ROM/SARAM	DARAM	External
DARAM	n	n	n+p
SARAM	n, n+1†	n	n+p
External	2n–1+(n–1)d	2n–1+(n–1)d	2n+2+nd+p
MMR◊	n	n	n+p

† Operand and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 16A

1 word, 1 cycle. Single data-memory (Smem) read operand or MMR read operand, and a stack-memory write operand.

Mnemonics

PSHD   PSHM

Cycles

Cycles for a Single Execution

Operand		Program		
Smem	**Stack**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1, 2†	1+p
	External	1	1, 2†	4+d+p
SARAM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†, 3‡	1	1+p
	External	1, 2†	1	4+d+p
DROM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†	1	1+p
	External	1, 2†	1	4+d+p
External	DARAM	1+d	1+d	2+d+p
	SARAM	1+d, 2+d†	1+d	2+d+p
	External	1+d	1+d	5+2d+p
MMR◊	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1	1+p
	External	1	1	4+d+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	**Stack**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n	n, n+1†	n+p
	SARAM	n, n+1†	n, n+1†	n+p
	External	2n–1+(n–1)d	2n–1+(n–1)d, 2n+(n–1)d†	2n+2+nd+p
SARAM	DARAM	n, n+1†	n	n+p
	SARAM	n, n+1†, 2n#, 2n+1‡	n, 2n#	n+p, 2n+p#
	External	2n–1+(n–1)d, 2n+(n–1)d†	2n–1+(n–1)d, 2n+(n–1)d†	2n+2+nd+p
DROM	DARAM	n, n+1†	n, n+1†	n+p
	SARAM	n, n+1†	n	n+p
	External	2n–1+(n–1)d, 2n+(n–1)d†	2n–1+(n–1)d	2n+2+nd+p
External	DARAM	n+nd	n+nd	n+1+nd+p
	SARAM	n+nd, n+1+nd†	n+nd	n+1+nd+p
	External	4n–3+(2n–1)d	4n–3+(2n–1)d	4n+1+2nd+p
MMR◊	DARAM	n	n, 2n†	n+p
	SARAM	n, n+1†	n	n+p
	External	2n–1+(n–1)d	2n–1+(n–1)d	2n+2+nd+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
# Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 16B 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and a stack-memory write operand.

Mnemonics PSHD

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	**Stack**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2, 3†	2+2p
	External	2	2, 3†	5+d+2p
SARAM	DARAM	2, 3†	2	2+2p
	SARAM	2, 3†, 4‡	2	2+2p
	External	2, 3†	2	5+d+2p
DROM	DARAM	2, 3†	2	2+2p
	SARAM	2, 3†	2	2+2p
	External	2, 3†	2	5+d+2p
External	DARAM	2+d	2+d	3+d+2p
	SARAM	2+d, 3+d†	2+d	3+d+2p
	External	2+d	2+d	6+2d+2p
MMR◊	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2	2+2p
	External	2	2	5+d+2p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 17A

1 word, 1 cycle. Single data-memory (Smem) write operand or MMR write operand, and a stack-memory read operand.

Mnemonics

POPD POPM

Cycles

Cycles for a Single Execution

Operand		Program		
Smem	**Stack**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1	1+p
	DROM	1, 2†	1	1+p
	External	1+d	1+d	2+d+p
	MMR◊	1	1, 2†	1+p
SARAM	DARAM	1, 2†	1, 2†	1+p
	SARAM	1, 2†, 3‡	1	1+p
	DROM	1, 2†	1	1+p
	External	1+d, 2+d†	1+d	2+d+p
	MMR◊	1, 2†	1	1+p
External	DARAM	1	1, 2†	4+d+p
	SARAM	1, 2†	1	4+d+p
	DROM	1, 2†	1	4+d+p
	External	1+d	1+d	5+2d+p
	MMR◊	1	1	4+d+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	**Stack**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n	n, n+1†	n+p
	SARAM	n, n+1†	n	n+p
	DROM	n, n+1†	n, n+1†	n+p
	External	n+nd	n+nd	n+1+nd+p
	MMR◊	n	n, 2n†	n+p
SARAM	DARAM	n, n+1†	n, n+1†	n+p
	SARAM	n, n+1†, 2n 2n+1‡	n, 2n	n+p, 2n+p
	DROM	n, n+1†	n	n+p
	External	n+nd, n+1+nd†	n+nd	n+1+nd+p
	MMR◊	n, n+1†	n	n+p
External	DARAM	2n–1+(n–1)d	2n–1+(n–1)d, 2n+(n–1)d†	2n+2+nd+p
	SARAM	2n–1+(n–1)d, 2n+(n–1)d†	2n–1+(n–1)d, 2n+(n–1)d†	2n+2+nd+p
	DROM	2n–1+(n–1)d, 2n+(n–1)d†	2n–1+(n–1)d	2n+2+nd+p
	External	4n–3+((2n–1)d	4n–3+(2n–1)d	4n+1+2nd+p
	MMR◊	2n–1+(n–1)d	2n–1+(n–1)d	2n+2+nd+p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 17B

2 words, 2 cycles. Single data-memory (Smem) write operand using long-offset indirect addressing, and a stack-memory read operand.

Mnemonics

POPD

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	**Stack**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2	2+2p
	DROM	2, 3†	2	2+2p
	External	2+d	2+d	3+d+2p
	MMR◊	2	2, 3†	2+2p
SARAM	DARAM	2, 3†	2, 3†	2+2p
	SARAM	2, 3†, 4‡	2	2+2p
	DROM	2, 3†	2	2+2p
	External	2+d, 3+d†	2+d	3+d+2p
	MMR◊	2, 3†	2	2+2p
External	DARAM	2	2, 3†	5+d+2p
	SARAM	2, 3†	2	5+d+2p
	DROM	2, 3†	2	5+d+2p
	External	2+d	2+d	6+2d+2p
	MMR◊	2	2	5+d+2p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 18A

2 words, 2 cycles. Single data-memory (Smem) read and write operand.

Mnemonics

ADDM ANDM ORM XORM

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	2	2, 3†	2+2p
SARAM	2, 4†	2	2+2p
External	2+d	2+d	6+2d+2p
MMR◊	2	2	2+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 18B

3 words, 3 cycles. Single data-memory (Smem) read and write operand using long-offset indirect addressing.

Mnemonics

ADDM ANDM ORM XORM

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	3	3, 4†	3+3p
SARAM	3, 5†	3	3+3p
External	3+d	3+d	7+2d+3p
MMR◊	3	3	3+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 19A

2 words, 2 cycles. Single data-memory (Smem) read operand or MMR read operand, and single data-memory (dmad) write operand; or single data-memory (dmad) read operand, and single data-memory (Smem) write operand or MMR write operand.

Mnemonics

MVDK MVDM MVKD MVMD

Cycles

Cycles for a Single Execution

Operand		Program		
Smem	**dmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2, 3†	2+2p
	External	2	2, 3†	5+d+2p
	MMR◊	2	2	2+2p
SARAM	DARAM	2, 3†	2	2+2p
	SARAM	2, 3†, 4‡	2	2+2p
	External	2, 3†	2	5+d+2p
	MMR◊	2, 3†	2	2+2p
DROM	DARAM	2, 3‡	2	2+2p
	SARAM	2, 3†	2	2+2p
	External	2, 3†	2	5+d+2p
	MMR◊	2, 3†	2	2+2p
External	DARAM	2+d	2+d	3+d+2p
	SARAM	2+d, 3+d†	2+d	3+d+2p
	External	2+d	2+d	6+2d+p
	MMR◊	2+d	2+d	3+d+2p
MMR◊	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2	2+2p
	External	2	2	5+d+2p
	MMR◊	2	2	2+2p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	**dmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+1	n+1, n+2†	n+1+2p
	SARAM	n+1, n+2†	n+1, n+2†	n+1+2p
	External	2n+(n–1)d	2n+(n–1)d, 2n+1+(n–1)d†	2n+3+nd+2p
	MMR◊	n+1	n+1	n+1+2p
SARAM	DARAM	n+1, n+2†	n+1	n+1+2p
	SARAM	2n, 2n+1†, 2n+2‡	2n	2n+2p
	External	2n+(n–1)d, 2n+1+(n–1)d†	2n+(n–1)d	2n+3+nd+2p
	MMR◊	n+1, n+2†	n+1	n+1+2p
DROM	DARAM	n+1, n+2†	n+1	n+1+2p
	SARAM	n+1, n+2†	n+1	n+1+2p
	External	2n+(n–1)d, 2n+1+(n–1)d†	2n+(n–1)d	2n+3+nd+2p
	MMR◊	n+1, n+2†	n+1	n+1+2p
External	DARAM	n+1+nd	n+1+nd	n+1+nd+2p
	SARAM	n+1+nd, n+2nd†	n+1+nd	n+1+nd+2p
	External	4n–2+(2n–1)d	4n–2+(2n–1)d	4n+2+2nd+2p
	MMR◊	n+1+nd	n+1+nd	n+1+nd+2p
MMR◊	DARAM	n+1	n+1	n+1+2p
	SARAM	n+1, n+2†	n+1	n+1+2p
	External	2n+(n–1)d	2n+(n–1)d	2n+3+nd+2p
	MMR◊	n+1	n+1	n+1+2p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 19B 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single data-memory (dmad) write operand, or single data-memory (dmad) read operand and single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics MVDK MVKD

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	**dmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	3	3, 4†	3+3p
	SARAM	3, 4†	3, 4†	3+3p
	External	3	3, 4†	6+d+3p
	MMR◊	3	3	3+3p
SARAM	DARAM	3, 4†	3	3+3p
	SARAM	3, 4†, 5‡	3	3+3p
	External	3, 4†	3	6+d+3p
	MMR◊	3, 4†	3	3+3p
DROM	DARAM	3, 4‡	3	3+3p
	SARAM	3, 4†	3	3+3p
	External	3, 4†	3	6+d+3p
	MMR◊	3, 4†	3	3+3p
External	DARAM	3+d	3+d	4+d+3p
	SARAM	3+d, 4+d†	3+d	4+d+3p
	External	3+d	3+d	7+2d+2p
	MMR◊	3+d	3+d	4+d+3p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Single Execution With Long-Offset Modifier (Continued)

Operand		Program		
Smem	**dmad**	**ROM/SARAM**	**DARAM**	**External**
MMR◊	DARAM	3	3, 4†	3+3p
	SARAM	3, 4†	3	3+3p
	External	3	3	6+d+3p
	MMR◊	3	3	3+3p

† Operand and code in same memory block
‡ Two operands and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 20A

2 words, 4 cycles. Single data-memory (Smem) read operand and single program-memory (pmad) write operand.

Mnemonics MVDP

Cycles

Cycles for a Single Execution

Operand		Program		
Smem	**pmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	4	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p
SARAM	DARAM	4, 5†	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p
DROM	DARAM	4, 5†	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p
External	DARAM	4+d	4+d	4+d+2p
	SARAM	4+d	4+d	4+d+2p
	External	4+d+pd	4+d+pd	6+d+pd+2p
MMR◊	DARAM	4	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		**Program**		
Smem	**pmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+3	n+3	n+3+2p
	SARAM	n+3	n+3	n+3+2p
	External	2n+2+(n–1)pd	2n+2+(n–1)pd	2n+4+npd+2p
SARAM	DARAM	n+3	n+3	n+3+2p
	SARAM	n+3, 2n+2#	n+3, 2n+2#	n+3+2p, 2n+2+2p#
	External	2n+2+(n–1)pd	2n+2+(n–1)pd	2n+4+npd+2p
DROM	DARAM	n+3	n+3	n+3+2p
	SARAM	n+3	n+3	n+3+2p
	External	2n+2+(n–1)pd	2n+2+(n–1)pd	2n+4+npd+2p
External	DARAM	n+3+npd	n+3+npd	n+3+npd+2p
	SARAM	n+3+npd	n+3+npd	n+3+npd+2p
	External	4n+nd+npd	4n+nd+npd	4n+2+nd+npd+2p
MMR◊	DARAM	n+3	n+3	n+3+2p
	SARAM	n+3	n+3	n+3+2p
	External	2n+2+(n–1)pd	2n+2+(n–1)pd	2n+4+npd+2p

Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 20B

3 words, 5 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single program-memory (pmad) write operand.

Mnemonics

MVDP

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	**pmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	5	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+2pd+3p
SARAM	DARAM	5, 6†	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+2pd+3p
DROM	DARAM	5, 6†	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+2pd+3p
External	DARAM	5+d	5+d	5+d+3p
	SARAM	5+d	5+d	5+d+3p
	External	5+d+2pd	5+d+2pd	7+d+2pd+3p
MMR◊	DARAM	5	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+3pd+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 21A

2 words, 3 cycles. Single program-memory (pmad) read operand and single data-memory (Smem) write operand.

Mnemonics

MVPD

Cycles

Cycles for a Single Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	3	3	3+2p
	SARAM	3	3	3+2p
	External	3	3	6+d+2p
	MMR◊	3	3	3+2p
SARAM	DARAM	3	3	3+2p
	SARAM	3	3	3+2p
	External	3	3	6+d+2p
	MMR◊	3	3	3+2p
PROM	DARAM	3	3	3+2p
	SARAM	3	3	3+2p
	External	3	3	6+d+2p
	MMR◊	3	3	3+2p
External	DARAM	3+pd	3+pd	3+pd+2p
	SARAM	3+pd	3+pd	3+pd+2p
	External	3+pd	3+pd	6+d+pd+2p
	MMR◊	3+pd	3+pd	3+pd+2p

◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2	n+2	n+2+2p
	External	2n+1+(n–1)d	2n+1+(n–1)d	2n+4+nd+2p
	MMR◊	n+2	n+2	n+2+2p
SARAM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2, 2n+1#	n+2, 2n+1#	n+2+2p
	External	2n+1+(n–1)d	2n+1+(n–1)d	2n+4+nd+2p
	MMR◊	n+2	n+2	n+2+2p
PROM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2	n+2	n+2+2p
	External	2n+1+(n–1)d	2n+1+(n–1)d	2n+4+nd+2p
	MMR◊	n+2	n+2	n+2+2p
External	DARAM	n+2+npd	n+2+npd	n+2+npd+2p
	SARAM	n+2+npd	n+2+npd	n+2+npd+2p
	External	4n–1+(n–1)d +npd	4n–1+(n–1)d +npd	4n+2+nd+npd+2p
	MMR◊	n+2+npd	n+2+npd	n+2+npd+2p

Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 21B 3 words, 4 cycles. Single program-memory (pmad) read operand and single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics MVPD

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	4	4	4+3p
	SARAM	4	4	4+3p
	External	4	4	7+d+3p
	MMR◊	4	4	4+3p
SARAM	DARAM	4	4	4+3p
	SARAM	4	4	4+3p
	External	4	4	7+d+3p
	MMR◊	4	4	4+3p
PROM	DARAM	4	4	4+3p
	SARAM	4	4	4+3p
	External	4	4	7+d+3p
	MMR◊	4	4	4+3p
External	DARAM	4+2pd	4+2pd	4+2pd+3p
	SARAM	4+2pd	4+2pd	4+2pd+3p
	External	4+2pd	4+2pd	7+d+2pd+3p
	MMR◊	4+2pd	4+2pd	4+2pd+3p

◊ Add one cycle for peripheral memory-mapped access.

Class 22A

2 words, 3 cycles. Single data-memory (Smem) read operand and single program-memory (pmad) read operand.

Mnemonics

MACP

Cycles

Cycles for a Single Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	3	3, 4†	3+2p
	SARAM	3, 4†	3	3+2p
	External	3+d	3+d	4+d+2p
	MMR◊	3	3	3+2p
SARAM	DARAM	3	3, 4†	3+2p
	SARAM	3, 4†	3	3+2p
	External	3+d	3+d	4+d+2p
	MMR◊	3	3	3+2p
PROM	DARAM	3	3, 4†	3+2p
	SARAM	3, 4†	3	3+2p
	External	3+d	3+d	4+d+2p
	MMR◊	3	3	3+2p
External	DARAM	3+pd	3+pd, 4+pd†	3+pd+2p
	SARAM	3+pd	3+pd	4+pd+2p
	External	4+d+pd	4+d+pd	4+d+pd+2p
	MMR◊	3+pd	3+pd	3+pd+2p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+2	n+2, n+3†	n+2+2p
	SARAM	n+2, n+3†	n+2	n+2+2p
	External	n+2+nd	n+2+nd	n+2+nd+2p
	MMR◊	n+2	n+2	n+2+2p
SARAM	DARAM	n+2	n+2, n+3†	n+2+2p
	SARAM	n+2, n+3†, 2n+2#	n+2, 2n+2#	n+2+2p, 2n+2+2p#
	External	n+2+nd	n+2+nd	n+2+nd+2p
	MMR◊	n+2	n+2	n+2+2p
PROM	DARAM	n+2	n+2, n+3†	n+2+2p
	SARAM	n+2, n+3†	n+2	n+2+2p
	External	n+2+nd	n+2+nd	n+2+nd+2p
	MMR◊	n+2	n+2	n+2+2p
External	DARAM	n+2+npd	n+2+npd, n+3+npd†	n+2+npd+2p
	SARAM	n+2+npd	n+2+npd	n+3+npd+2p
	External	2n+2+nd+npd	2n+2+nd+npd	2n+2+nd+npd +2p
	MMR◊	n+2+npd	n+2+npd	n+2+npd+2p

† Operand and code in same memory block
# Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 22B 3 words, 4 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single program-memory (pmad) read operand.

Mnemonics MACP

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	4	4, 5†	4+3p
	SARAM	4, 5†	4	4+3p
	External	4+d	4+d	5+d+3p
	MMR◊	4	4	4+3p
SARAM	DARAM	4	4, 5†	4+3p
	SARAM	4, 5†	4	4+3p
	External	4+d	4+d	5+d+3p
	MMR◊	4	4	4+3p
PROM	DARAM	4	4, 5†	4+3p
	SARAM	4, 5†	4	4+3p
	External	4+d	4+d	5+d+3p
	MMR◊	4	4	4+3p
External	DARAM	4+2pd	4+2pd, 5+2pd†	4+2pd+3p
	SARAM	4+2pd	4+2pd	5+2pd+3p
	External	5+d+2pd	5+d+2pd	5+d+2pd+3p
	MMR◊	4+2pd	4+2pd	4+2pd+3p

† Operand and code in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 23A 2 words, 3 cycles. Single data-memory (Smem) read operand, single data-memory (Smem) write operand, and single program-memory (pmad) read operand.

Mnemonics MACD

Cycles

Cycles for a Single Execution

Operand		**Program**		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	3, 4#	3, 4#	3+2p, 4+2p#
	SARAM	3, 4†	3, 4†	3+2p
	External	3+d	3+d	6+2d+2p
	MMR◊	3	3	3+2p
SARAM	DARAM	3, 4†	3	3+2p
	SARAM	3, 4#	3, 4#	3+2p, 4+2p#
	External	3+d	3+d	6+2d+2p
	MMR◊	3	3	3+2p
PROM	DARAM	3	3	3+2p
	SARAM	3, 4†	3	3+2p
	External	3+d	3+d	6+2d+2p
	MMR◊	3	3	3+2p
External	DARAM	3+pd	3+pd	3+pd+2p
	SARAM	3+pd	3+pd	3+pd+2p
	External	4+d+pd	4+d+pd	7+d+pd+2p
	MMR◊	3+pd	3+pd	4+pd+2p

† Operand and code in same memory block
# Two operands in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+2, 2n+2#	n+2, 2n+2#	n+2+2p, 2n+2+2p#
	SARAM	n+2, n+3†	n+2, n+3†	n+2+2p
	External	4n+1+2nd	4n+1+2nd	4n+2+2nd+2p
	MMR◊	n+2	n+2	n+2+2p
SARAM	DARAM	n+2, n+3†	n+2	n+2+2p
	SARAM	n+2, 2n+2#	n+2, 2n+2#	n+2+2p, 2n+2+2p#
	External	4n+1+2nd	4n+1+2nd	4n+2+2nd+2p
	MMR◊	n+2	n+2	n+2+2p
PROM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2, n+3†	n+2	n+2+2p
	External	4n+1+2nd	4n+1+2nd	4n+2+2nd+2p
	MMR◊	n+2	n+2	n+2+2p
External	DARAM	n+2+npd	n+2+npd, n+3+npd†	n+2+npd+2p
	SARAM	n+2+npd	n+2+npd	n+2+npd+2p
	External	5n–1+nd+npd	5n–1+nd+npd	5n+2+nd+npd +2p
	MMR◊	n+2+npd	n+2+npd	4n+3+npd+2p

† Operand and code in same memory block
# Two operands in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 23B 3 words, 4 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing, single data-memory (Smem) write operand using long-offset indirect addressing, and single program-memory (pmad) read operand.

Mnemonics MACD

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	4, 5#	4, 5#	4+3p, 5+3p#
	SARAM	4, 5†	4, 5†	4+3p
	External	4+d	4+d	7+2d+3p
	MMR◊	4	4	4+3p
SARAM	DARAM	4, 5†	4	4+3p
	SARAM	4, 5#	4, 5#	4+3p, 5+3p#
	External	4+d	4+d	7+2d+3p
	MMR◊	4	4	4+3p
PROM	DARAM	4	4	4+3p
	SARAM	4, 5†	4	4+3p
	External	4+d	4+d	7+2d+3p
	MMR◊	4	4	4+3p
External	DARAM	4+2pd	4+2pd	4+pd+3p
	SARAM	4+2pd	4+2pd	4+2pd+3p
	External	5+d+2pd	5+d+2pd	8+d+2pd+3p
	MMR◊	4+2pd	4+2pd	5+2pd+3p

† Operand and code in same memory block
# Two operands in same memory block
◊ Add one cycle for peripheral memory-mapped access.

Class 24A 1 word, 1 cycle. Single data-memory (Smem) read operand and single data-memory (Smem) write operand.

Mnemonics DELAY LTD

Cycles

Cycles for a Single Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	1	1, 2†	1+p
SARAM	1, 3†	1	1+p
External	1+d	1+d	5+p+2d

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	n	n, n+1†	n+p
SARAM	2n–1, 2n+1†	2n–1	2n–1+p
External	4n–3+(2n–1)d	4n–3+(2n–1)d	4n+1+p+2nd

† Operand and code in same memory block

Class 24B 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics DELAY LTD

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2, 3†	2+2p
SARAM	2, 4†	2	2+2p
External	2+d	2+d	6+2p+2d

† Operand and code in same memory block

Class 25A

1 word, 5 cycles. Single program-memory (pmad) read address and single data-memory (Smem) write operand.

Mnemonics

READA

Cycles

Cycles for a Single Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	8+d+p
	MMR◊	5	5	5+p
SARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	8+d+p
	MMR◊	5	5	5+p
PROM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	8+d+p
	MMR◊	5	5	5+p
External	DARAM	5+pd	5+pd	5+pd+p
	SARAM	5+pd	5+pd	5+pd+p
	External	5+pd	5+pd	8+pd+d+p
	MMR◊	5+pd	5+pd	5+pd+p

◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n–1)d	2n+3+(n–1)d	2n+6+nd+np
	MMR◊	n+4	n+4	n+4+p
SARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4, 2n+3#	n+4, 2n+3#	n+4+p, 2n+3+p#
	External	2n+3+(n–1)d	2n+3+(n–1)d	2n+6+nd+p
	MMR◊	n+4	n+4	n+4+p
PROM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n–1)d	2n+3+(n–1)d	2n+6+nd+p
	MMR◊	n+4	n+4	n+4+p
External	DARAM	n+4+npd	n+4+npd	n+4+npd+p
	SARAM	n+4+npd	n+4+npd	n+4+npd+p
	External	4n+1+(n–1)d +npd	4n+1+(n–1)d +npd	4n+4+nd+npd +p
	MMR◊	n+4+npd	n+4+npd	n+4+npd+p

Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 25B

2 words, 6 cycles. Single program-memory (pmad) read address and single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics

READA

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
pmad	**Smem**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	9+d+2p
	MMR◊	6	6	6+2p
SARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	9+d+2p
	MMR◊	6	6	6+2p
PROM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	9+d+2p
	MMR◊	6	6	6+2p
External	DARAM	6+2pd	6+2pd	6+2pd+2p
	SARAM	6+2pd	6+2pd	6+2pd+2p
	External	6+2pd	6+2pd	9+2pd+d+2p
	MMR◊	6+2pd	6+2pd	6+2pd+2p

◊ Add one cycle for peripheral memory-mapped access.

Class 26A 1 word, 5 cycles. Single data-memory (Smem) read operand and single program-memory (pmad) write address.

Mnemonics WRITA

Cycles

Cycles for a Single Execution

Operand		Program		
Smem	**pmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p
SARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p
DROM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p
External	DARAM	5+pd	5+pd	5+pd+p
	SARAM	5+pd	5+pd	5+pd+p
	External	5+d	5+d	7+d+pd+p
MMR◊	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p

◊ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	**pmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n–1)pd	2n+3+(n–1)pd	2n+3+npd+p
SARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4, 2n+3#	n+4, 2n+3#	n+4+p, 2n+3+p#
	External	2n+3+(n–1)pd	2n+3+(n–1)pd	2n+3+npd+p
DROM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n–1)pd	2n+3+(n–1)pd	2n+3+npd+p
External	DARAM	n+4+npd	n+4+npd	n+4+npd+p
	SARAM	n+4+npd	n+4+npd	n+4+npd+p
	External	4n+1+nd +(n–1)pd	4n+1+nd +(n–1)pd	4n+3+nd+npd +p
MMR◊	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n–1)pd	2n+3+(n–1)pd	2n+3+npd+p

Two operands in same memory block
◊ Add n cycles for peripheral memory-mapped access.

Class 26B 2 words, 6 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single program-memory (pmad) write address.

Mnemonics WRITA

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	**pmad**	**ROM/SARAM**	**DARAM**	**External**
DARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p
SARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p
DROM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p
External	DARAM	6+2pd	6+2pd	6+2pd+2p
	SARAM	6+2pd	6+2pd	6+2pd+2p
	External	6+d	6+d	8+d+2pd+2p
MMR◊	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p

◊ Add one cycle for peripheral memory-mapped access.

Class 27A 2 words, 2 cycles. Single I/O port read operand and single data-memory (Smem) write operand.

Mnemonics PORTR

Cycles

Cycles for a Single Execution

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	3+io	3+io	6+2p+io
	SARAM	3+io, 4+io†	3+io	6+2p+io
	External	3+io	3+io	9+2p+d+io

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	2n+1+nio	2n+1+nio	2n+4+2p+nio
	SARAM	2n+1+nio, 2n+2+nio†	2n+1+nio	2n+4+2p+nio
	External	5n–2+nio +(n–1)d	5n–2+nio +(n–1)d	5n+4+2p +nio+nd

† Operand and code in same memory block

Class 27B 3 words, 3 cycles. Single I/O port read operand and single data-memory (Smem) write operand using long-offset indirect addressing.

Mnemonics PORTR

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	4+io	4+io	7+3p+io
	SARAM	4+io, 5+io†	4+io	7+3p+io
	External	4+io	4+io	10+3p+d+io

† Operand and code in same memory block

Class 28A

2 words, 2 cycles. Single data-memory (Smem) read operand and single I/O port write operand.

Mnemonics

PORTW

Cycles

Cycles for a Single Execution

Operand		Program		
Port	**Smem**	**ROM/SARAM**	**DARAM**	**External**
External	DARAM	2	2, 3†	6+2p+io
	SARAM	2, 3†	2	6+2p+io
	DROM	2, 3†	2	6+2p+io
	External	2+d	2+d	7+2p+d+io

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand		Program		
Port	**Smem**	**ROM/SARAM**	**DARAM**	**External**
External	DARAM	2n+(n–1)io	2n+(n–1)io, 2n+1+(n–1)io†	2n+4+2p+nio
	SARAM	2n+(n–1)io, 2n+1+(n–1)io†	2n+(n–1)io	2n+4+2p+nio
	DROM	2n+(n–1)io, 2n+1+(n–1)io†	2n+(n–1)io	2n+4+2p+nio
	External	5n–3+nd +(n–1)io	5n–3+nd +(n–1)io	5n+2+2p+nd +nio

† Operand and code in same memory block

Class 28B

3 words, 3 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single I/O port write operand.

Mnemonics

PORTW

Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Port	**Smem**	**ROM/SARAM**	**DARAM**	**External**
External	DARAM	3	3, 4†	7+3p+io
	SARAM	3, 4†	3	7+3p+io
	DROM	3, 4†	3	7+3p+io
	External	3+d	3+d	8+3p+d+io

† Operand and code in same memory block

Class 29A 2 words, 4 cycles, 2 cycles (delayed), 2 cycles (false condition). Single program-memory (pmad) operand.

Mnemonics B[D] BANZ[D] FB[D] RPTB[D]

Cycles

Cycles for a Single Execution

Program		
ROM/SARAM	**DARAM**	**External**
4	4	4+4p

Cycles for a Single Delayed Execution

Program		
ROM/SARAM	**DARAM**	**External**
2	2	2+2p

Class 29B 2 words, 4 cycles, 2 cycles (delayed). Single program-memory (pmad) operand.

Mnemonics CALL[D] FCALL[D]

Cycles

Cycles for a Single Execution

Operand	**Program**		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	4	4	4+4p
SARAM	4, 5†	4	4+4p
External	4	4	7+4p+d

† Operand and code in same memory block

Cycles for a Single Delayed Execution

Operand	**Program**		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	2	2	2+2p
SARAM	2, 3†	2	2+2p
External	2	2	5+2p+d

† Operand and code in same memory block

Class 30A

1 word, 6 cycles, 4 cycles (delayed). Single register operand.

Mnemonics

BACC[D] FBACC[D]

Cycles

Cycles for a Single Execution

Program		
ROM/SARAM	**DARAM**	**External**
6	6	6+3p

Cycles for a Single Delayed Execution

Program		
ROM/SARAM	**DARAM**	**External**
4	4	4+p

Class 30B

1 word, 6 cycles, 4 cycles (delayed). Single register operand.

Mnemonics

CALA[D] FCALA[D]

Cycles

Cycles for a Single Execution

	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	6	6	6+3p
SARAM	6	6	6+3p
External	6	6	7+3p+d

Cycles for a Single Delayed Execution

	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	4	4	4+p
SARAM	4	4	4+p
External	4	4	5+p+d

Class 31A

2 words, 5 cycles, 3 cycles (delayed). Single program-memory (pmad) operand and short-immediate operands.

Mnemonics BC[D]

Cycles

Cycles for a Single Execution

	Program		
Condition	**ROM/SARAM**	**DARAM**	**External**
True	5	5	5+4p
False	3	3	3+2p

Cycles for a Single Delayed Execution

	Program		
Condition	**ROM/SARAM**	**DARAM**	**External**
True	3	3	3+2p
False	3	3	3+2p

Class 31B 2 words, 5 cycles, 3 cycles (delayed), 3 cycles (false condition). Single program-memory (pmad) operand and short-immediate operands.

Mnemonics CC[D]

Cycles

Cycles for a Single True Condition Execution

Operand	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	5	5	5+4p
SARAM	5, 6†	5	5+4p
External	5	5	8+4p+d

† Operand and code in same memory block

Cycles for a Single False Condition Execution

Operand	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3	3+2p
SARAM	3, 4†	3	3+2p
External	3	3	6+2p+d

† Operand and code in same memory block

Cycles for a Single Delayed Execution

Operand	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3	3+2p
SARAM	3, 4†	3	3+2p
External	3	3	6+2p+d

† Operand and code in same memory block

Class 32

1 word, 5 cycles, 3 cycles (delayed), 3 cycles (false condition). No operand, or short-immediate operands.

Mnemonics

RC[D] RET[D] RETE[D]

Cycles

Cycles for a Single Execution

Operand	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	5	5, 6†	5+3p
SARAM	5, 6†	5	5+3p
External	5+d	5+d	6+d+3p

† Operand and code in same memory block

Cycles for a Single Delayed Execution

Operand	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	3	3, 4†	3+p
SARAM	3, 4†	3	3+p
External	3+d	3+d	4+d+p

† Operand and code in same memory block

Class 33 1 word, 3 cycles, 1 cycle (delayed). No operand.

Mnemonics RETF[D]

Cycles

Cycles for a Single Execution

Program		
ROM/SARAM	**DARAM**	**External**
3	3	3+p

Cycles for a Single Delayed Execution

Program		
ROM/SARAM	**DARAM**	**External**
1	1	1+p

Class 34 1 word, 6 cycles, 4 cycles (delayed). No operand.

Mnemonics FRET[D] FRETE[D]

Cycles

Cycles for a Single Execution

	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	6	6, 8†	6+3p
SARAM	6, 8†	6	6+3p
External	6+2d	6+2d	8+3p+d

† Operand and code in same memory block

Cycles for a Single Delayed Execution

	Program		
Stack	**ROM/SARAM**	**DARAM**	**External**
DARAM	4	4, 6†	4+p
SARAM	4, 6†	4	4+p
External	4+2d	4+2d	6+p+2d

† Operand and code in same memory block

Class 35

1 word, 3 cycles. No operand or single short-immediate operand.

Mnemonics

INTR RESET TRAP

Cycles

Cycles for a Single Execution

Program		
ROM/SARAM	**DARAM**	**External**
3	3	3+p

Class 36

1 word, 4 cycles (minimum). Single short-immediate operand.

Mnemonics

IDLE

Cycles

The number of cycles needed to execute this instruction depends on the idle period.

Chapter 4

Assembly Language Instructions

This section provides detailed information on the instruction set for the TMS320C54x™ DSP family. The C54x™ DSP instruction set supports numerically intensive signal-processing operations as well as general-purpose applications, such as multiprocessing and high-speed control.

See Section 1.1, *Instruction Set Symbols and Abbreviations*, for definitions of symbols and abbreviations used in the description of assembly language instructions. See Section 1.2, *Example Description of Instruction*, for a description of the elements in an instruction. See Chapter 2 for a summary of the instruction set.

Syntax **ABDST** *Xmem*, *Ymem*

Operands Xmem, Ymem: Dual data-memory operands

Opcode

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

Execution

(B) + |(A(32–16))| → B
((Xmem) − (Ymem)) << 16 → A

Status Bits

Affected by OVM, FRCT, and SXM
Affects C, OVA, and OVB

Description This instruction calculates the absolute value of the distance between two vectors, *Xmem* and *Ymem*. The absolute value of accumulator A(32–16) is added to accumulator B. The content of *Ymem* is subtracted from *Xmem*, and the result is left-shifted 16 bits and stored in accumulator A. If the fractional mode bit is logical 1 (FRCT = 1), the absolute value is multiplied by 2.

Words 1 word

Cycles 1 cycle

Classes Class 7 (see page 3-12)

Example

```
ABDST *AR3+, *AR4+
```

	Before Instruction	After Instruction
A	FF ABCD 0000	FF FFAB 0000
B	00 0000 0000	00 0000 5433
AR3	0100	0101
AR4	0200	0201
FRCT	0	0
Data Memory		
0100h	0055	0055
0200h	00AA	00AA

Syntax **ABS** *src* [, *dst*]

Operands src, dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution |(src)| → dst (or src if dst is not specified)

Status Bits OVM affects this instruction as follows:

If OVM = 1, the absolute value of 80 0000 0000h is 00 7FFF FFFFh.
If OVM = 0, the absolute value of 80 0000 0000h is 80 0000 0000h.

Affects C and OVdst (or OVsrc, if dst = src)

Description This instruction calculates the absolute value of *src* and loads the value into *dst.* If no *dst* is specified, the absolute value is loaded into *src.*

If the result of the operation is equal to 0, the carry bit, C, is set.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
ABS A, B
```

	Before Instruction			After Instruction	
A	FF FFFF FFCB	–53	A	FF FFFF FFCB	–53
B	FF FFFF FC18	–1000	B	00 0000 0035	+53

Example 2

```
ABS A
```

	Before Instruction		After Instruction
A	03 1234 5678	A	00 7FFF FFFF
OVM	1	OVM	1

Example 3

```
ABS A
```

	Before Instruction		After Instruction
A	03 1234 5678	A	03 1234 5678
OVM	0	OVM	0

Syntax

1: **ADD** *Smem, src*
2: **ADD** *Smem,* **TS***, src*
3: **ADD** *Smem,* **16**, *src* [, *dst*]
4: **ADD** *Smem* [, *SHIFT*], *src* [, *dst*]
5: **ADD** *Xmem, SHFT*, *src*
6: **ADD** *Xmem, Ymem, dst*
7: **ADD** *#lk* [, *SHFT*], *src* [, *dst*]
8: **ADD** *#lk*, **16**, *src* [, *dst*]
9: **ADD** *src* [, *SHIFT*], [, *dst*]
10: **ADD** *src,* **ASM** [, *dst*]

Operands

Smem: Single data-memory operand
Xmem, Ymem: Dual data-memory operands
src, dst: A (accumulator A)
B (accumulator B)

$-32\,768 \leq lk \leq 32\,767$
$-16 \leq SHIFT \leq 15$
$0 \leq SHFT \leq 15$

Opcode

1:

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

2:

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

3:

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

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	S	D	0	0	0	S	H	I	F	T

5:

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

6:

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

7:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	0	0	0	S	H	F	T
16-bit constant															

8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	0	0	0
16-bit constant															

9:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	0	0	S	H	I	F	T

10:

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

Execution

1: (Smem) + (src) → src
2: (Smem) << (TS) + (src) → src
3: (Smem) << 16 + (src) → dst
4: (Smem) [<< SHIFT] + (src) → dst
5: (Xmem) << SHFT + (src) → src
6: ((Xmem) + (Ymem)) << 16 → dst
7: lk << SHFT + (src)→ dst
8: lk << 16 + (src) → dst
9: (src or [dst]) + (src) << SHIFT → dst
10: (src or [dst]) + (src) << ASM → dst

Status Bits

Affected by SXM and OVM
Affects C and OVdst (or OVsrc, if dst = src)

For instruction syntax 3, if the result of the addition generates a carry, the carry bit, C, is set to 1; otherwise, C is not affected.

Description

This instruction adds a 16-bit value to the content of the selected accumulator or to a 16-bit operand *Xmem* in dual data-memory operand addressing mode. The 16-bit value added is one of the following:

- The content of a single data-memory operand (*Smem*)
- The content of a dual data-memory operand (*Ymem*)
- A 16-bit immediate operand (#*lk*)
- The shifted value in *src*

If *dst* is specified, this instruction stores the result in *dst*. If no *dst* is specified, this instruction stores the result in *src*. Most of the second operands can be shifted. For a left shift:

- Low-order bits are cleared
- High-order bits are:
  - Sign extended if SXM = 1
  - Cleared if SXM = 0

For a right shift, the high-order bits are:

- Sign extended if SXM = 1
- Cleared if SXM = 0

Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- Syntax 4: If *dst* = *src* and *SHIFT* = 0, then the instruction opcode is assembled as syntax 1.
- Syntax 4: If *dst* = *src*, *SHIFT* ≤ 15 and *Smem* indirect addressing mode is included in *Xmem*, then the instruction opcode is assembled as syntax 5.
- Syntax 5: If *SHIFT* = 0, the instruction opcode is assembled as syntax 1.

Words

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 word
Syntaxes 4, 7, and 8: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 cycle
Syntaxes 4, 7, and 8: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1, 2, 3, and 5: Class 3A (see page 3-5)
Syntaxes 1, 2, and 3: Class 3B (see page 3-6)
Syntax 4: Class 4A (see page 3-7)
Syntax 4: Class 4B (see page 3-8)
Syntax 6: Class 7 (see page 3-12)
Syntaxes 7 and 8: Class 2 (see page 3-4)
Syntaxes 9 and 10: Class 1 (see page 3-3)

Example 1

```
ADD *AR3+, 14, A
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0540 1200
C	1	C	0
AR3	0100	AR3	0101
SXM	1	SXM	1
Data Memory			
0100h	1500	0100h	1500

Example 2

```
ADD A, –8, B
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	00 0000 1812
C	1	C	0

Example 3

```
ADD #4568, 8, A, B
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	00 0045 7A00
C	1	C	0

Example 4

```
ADD *AR2+, *AR2–, A   ;after accessing the operands, AR2
                      ;is incremented by one.
```

Example 4 shows the same auxiliary register (AR2) with different addressing modes specified for both operands. The mode defined by the Xmod field (*AR2+) is used for addressing.

Syntax **ADDC** *Smem, src*

Operands
Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (Smem) + (src) + (C) → src

Status Bits
Affected by OVM, C
Affects C and OVsrc

Description This instruction adds the 16-bit single data-memory operand *Smem* and the value of the carry bit (C) to *src*. This instruction stores the result in *src*. Sign extension is suppressed regardless of the value of the SXM bit.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes
Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
ADDC *+AR2(5), A
```

	Before Instruction	After Instruction
A	00 0000 0013	00 0000 0018
C	1	0
AR2	0100	0105
Data Memory		
0105h	0004	0004

Syntax **ADDM** *#lk*, *Smem*

Operands Smem: Single data-memory operand
$-32\,768 \leq lk \leq 32\,767$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution #lk + (Smem) → Smem

Status Bits Affected by OVM and SXM
Affects C and OVA

Description This instruction adds the 16-bit single data-memory operand *Smem* to the 16-bit immediate memory value *lk* and stores the result in *Smem*.

Note:

This instruction is not repeatable.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 18A (see page 3-39)
Class 18B (see page 3-39)

Example 1

```
ADDM 0123Bh, *AR4+
```

	Before Instruction	After Instruction
AR4	0100	0101
Data Memory		
0100h	0004	123F

Example 2

```
ADDM 0FFF8h, *AR4+
```

	Before Instruction	After Instruction
OVM	1	1
SXM	1	1
AR4	0100	0101
Data Memory		
0100h	8007	8000

Syntax **ADDS** *Smem, src*

Operands
Smem: Single data-memory operands
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution uns(Smem) + (src) → src

Status Bits
Affected by OVM
Affects C and OVsrc

Description This instruction adds the 16-bit single data-memory operand *Smem* to *src* and stores the result in *src*. Sign extension is suppressed regardless of the value of the SXM bit.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes
Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
ADDS *AR2–, B
```

	Before Instruction		After Instruction
B	00 0000 0003	B	00 0000 F009
C	x	C	0
AR2	0100	AR2	00FF
Data Memory			
0104h	F006	0104h	F006

Syntax

1: **AND** *Smem*, *src*
2: **AND** *#lk* [, *SHFT*], *src* [, *dst*]
3: **AND** *#lk*, **16**, *src* [, *dst*]
4: **AND** *src* [, *SHIFT*], [, *dst*]

Operands

Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)
$-16 \leq \text{SHIFT} \leq 15$
$0 \leq \text{SHFT} \leq 15$
$0 \leq \text{lk} \leq 65\,535$

Opcode

1:

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

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	0	1	1	S	H	F	T
16-bit constant															

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	0	1	1
16-bit constant															

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	1	0	0	S	H	I	F	T

Execution

1: (Smem) AND (src) → src
2: lk << SHFT AND (src)→ dst
3: lk << 16 AND (src)→ dst
4: (dst) AND (src) << SHIFT → dst

Status Bits

None

Description

This instruction ANDs the following to *src*:

- ❑ A 16-bit operand *Smem*
- ❑ A 16-bit immediate operand *lk*
- ❑ The source or destination accumulator (*src* or *dst*)

If a shift is specified, this instruction left-shifts the operand before the AND. For a left shift, the low-order bits are cleared and the high-order bits are not sign extended. For a right shift, the high-order bits are not sign extended.

Words

Syntaxes 1 and 4: 1 word
Syntaxes 2 and 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1 and 4: 1 cycle
Syntaxes 2 and 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntaxes 2 and 3: Class 2 (see page 3-4)
Syntax 4: Class 1 (see page 3-3)

Example 1

```
AND *AR3+, A
```

	Before Instruction	After Instruction
A	00 00FF 1200	00 0000 1000
AR3	0100	0101
Data Memory		
0100h	1500	1500

Example 2

```
AND A, 3, B
```

	Before Instruction	After Instruction
A	00 0000 1200	00 0000 1200
B	00 0000 1800	00 0000 1000

Syntax **ANDM** *#lk*, *Smem*

Operands Smem: Single data-memory operand
$0 \leq lk \leq 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution lk AND (Smem) → Smem

Status Bits None

Description This instruction ANDs the 16-bit single data-memory operand *Smem* with a 16-bit long constant *lk*. The result is stored in the data-memory location specified by *Smem*.

Note:

This instruction is not repeatable.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 18A (see page 3-39)
Class 18B (see page 3-39)

Example 1

```
ANDM #00FFh, *AR4+
```

	Before Instruction	After Instruction
AR4	0100	0101
Data Memory		
0100h	0444	0044

Example 2

```
ANDM #0101h, 4; DP = 0
```

	Before Instruction	After Instruction
Data Memory		
0004h	00 0000 0100	00 0000 0100

Syntax **B**[**D**] *pmad*

Operands 0 ≤ pmad ≤ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	Z	0	0	1	1	1	0	0	1	1
16-bit constant															

Execution pmad → PC

Status Bits None

Description This instruction passes control to the designated program-memory address (*pmad*), which can be either a symbolic or numeric address. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable.

Words 2 words

Cycles 4 cycles
2 cycles (delayed)

Classes Class 29A (see page 3-66)

Example 1

```
B 2000h
```

	Before Instruction	After Instruction
PC	1F45	2000

Example 2

```
BD 1000h
ANDM 4444h, *AR1+
```

	Before Instruction	After Instruction
PC	1F45	1000

After the operand has been ANDed with 4444h, the program continues executing from location 1000h.

Syntax **BACC**[**D**] *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (src(15–0)) → PC

Status Bits None

Description This instruction passes control to the 16-bit address in the low part of *src* (bits 15–0). If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 6 cycles
4 cycles (delayed)

Classes Class 30A (see page 3-67)

Example 1

```
BACC A
```

	Before Instruction		After Instruction
A	00 0000 3000	A	00 0000 3000
PC	1F45	PC	3000

Example 2

```
BACCD B
ANDM 4444h, *AR1+
```

	Before Instruction		After Instruction
B	00 0000 2000	B	00 0000 2000
PC	1F45	PC	2000

After the operand has been ANDed with 4444h value, the program continues executing from location 2000h.

Syntax **BANZ**[**D**] *pmad*, *Sind*

Operands Sind: Single indirect addressing operand
$0 \leq$ pmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	Z	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution If ((ARx) $\neq$ 0)
Then
pmad $\rightarrow$ PC
Else
(PC) + 2 $\rightarrow$ PC

Status Bits None

Description This instruction branches to the specified program-memory address (*pmad*) if the value of the current auxiliary register ARx is not 0. Otherwise, the PC is incremented by 2. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable.

Words 2 words

Cycles 4 cycles (true condition)
2 cycles (false condition)
2 cycles (delayed)

Classes Class 29A (see page 3-66)

Example 1

```
BANZ 2000h, *AR3–
```

	Before Instruction	After Instruction
PC	1000	2000
AR3	0005	0004

Example 2

```
BANZ 2000h, *AR3–
```

	Before Instruction	After Instruction
PC	1000	1002
AR3	0000	FFFF

Example 3

```
BANZ 2000h, *AR3(–1)
```

	Before Instruction		After Instruction
PC	1000	PC	1003
AR3	0001	AR3	0001

Example 4

```
BANZD 2000h, *AR3–
ANDM 4444h, *AR5+
```

	Before Instruction		After Instruction
PC	1000	PC	2000
AR3	0004	AR3	0003

After the memory location has been ANDed with 4444h, the program continues executing from location 2000h.

Syntax **BC**[**D**] *pmad*, *cond* [, *cond* [, *cond*]]

Operands $0 \leq \text{pmad} \leq 65\,535$

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{\text{BIO}}$ low	0000 0011	NBIO	$\overline{\text{BIO}}$ high	0000 0010
C	C = 1	0000 1100	NC	C = 0	0000 1000
TC	TC = 1	0011 0000	NTC	TC = 0	0010 0000
AEQ	(A) = 0	0100 0101	BEQ	(B) = 0	0100 1101
ANEQ	(A) ≠ 0	0100 0100	BNEQ	(B) ≠ 0	0100 1100
AGT	(A) > 0	0100 0110	BGT	(B) > 0	0100 1110
AGEQ	(A) ≥ 0	0100 0010	BGEQ	(B) ≥ 0	0100 1010
ALT	(A) < 0	0100 0011	BLT	(B) < 0	0100 1011
ALEQ	(A) ≤ 0	0100 0111	BLEQ	(B) ≤ 0	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	Z	0	C	C	C	C	C	C	C	C
16-bit constant															

Execution If (cond(s))
Then
pmad → PC
Else
(PC) + 2 → PC

Status Bits Affects OVA or OVB if OV or NOV is chosen

Description This instruction branches to the program-memory address (*pmad*) if the specified condition(s) is met. The two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory. If the condition(s) is met, the two words following the instruction are flushed from the pipeline and execution begins at *pmad*. If the condition(s) is not met, the PC is incremented by 2 and the two words following the instruction are executed.

If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction is fetched from program memory and executed. The two words following the delayed instruction have no effect on the conditions being tested. If the condition(s) is met, execution continues at *pmad*. If the condition(s) is not met, the PC is incremented by 2 and the two words following the delayed instruction are executed.

This instruction tests multiple conditions before passing control to another section of the program. This instruction can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

Group1: You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time. The accumulator must be the same for both conditions; you cannot test conditions for both accumulators with the same instruction. For example, you can test AGT and AOV at the same time, but you cannot test AGT and BOV at the same time.

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

Conditions for This Instruction

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

Note:

This instruction is not repeatable.

Words 2 words

Cycles 5 cycles (true condition)
3 cycles (false condition)
3 cycles (delayed)

Classes Class 31A (see page 3-68)

Example 1

```
BC 2000h, AGT
```

	Before Instruction	After Instruction
A	00 0000 0053	00 0000 0053
PC	1000	2000

Example 2

```
BC 2000h, AGT
```

	Before Instruction	After Instruction
A	FF FFFF FFFF	FF FFFF FFFF
PC	1000	1002

Example 3

```
BCD 1000h, BOV
ANDM 4444h, *AR1+
```

	Before Instruction	After Instruction
PC	3000	1000
OVB	1	1

After the memory location is ANDed with 4444h, the branch is taken if the condition (OVB) is met. Otherwise, execution continues at the instruction following this instruction.

Example 4

```
BC 1000h, TC, NC, BIO
```

	Before Instruction	After Instruction
PC	3000	3002
C	1	1

Syntax **BIT** *Xmem*, *BITC*

Operands Xmem: Dual data-memory operand
$0 \leq$ BITC ≤ 15

Opcode

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

Execution (Xmem(15 – BITC)) → TC

Status Bits Affects TC

Description This instruction copies the specified bit of the dual data-memory operand *Xmem* into the TC bit of status register ST0. The following table lists the bit codes that correspond to each bit in data memory.

The bit code corresponds to BITC and the bit address corresponds to (15 – BITC).

Bit Codes for This Instruction

Bit Address	Bit Code	Bit Address	Bit Code
(LSB) 0	1111	8	0111
1	1110	9	0110
2	1101	10	0101
3	1100	11	0100
4	1011	12	0011
5	1010	13	0010
6	1001	14	0001
7	1000	(MSB) 15	0000

Words 1 word

Cycles 1 cycle

Classes Class 3A (see page 3-5)

Example

```
BIT *AR5+, 15-12; test bit 12
```

	Before Instruction		After Instruction
AR5	0100	AR5	0101
TC	0	TC	1
Data Memory			
0100h	7688	0100h	7688

Syntax **BITF** *Smem*, #*lk*

Operands Smem: Single data-memory operand
$0 \leq lk \leq 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution If ((Smem) AND lk) = 0
Then
  0 → TC
Else
  1 → TC

Status Bits Affects TC

Description This instruction tests the specified bit or bits of the data-memory value *Smem*. If the specified bit (or bits) is 0, the TC bit in status register ST0 is cleared to 0; otherwise, TC is set to 1. The *lk* constant is a mask for the bit or bits tested.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 6A (see page 3-10)
Class 6B (see page 3-11)

Example 1

```
BITF 5, 00FFh
```

	Before Instruction	After Instruction
TC	x	0
DP	004	004
Data Memory		
0205h	5400	5400

Example 2

```
BITF 5, 0800h
```

	Before Instruction	After Instruction
TC	x	1
DP	004	004
Data Memory		
0205h	0F7F	0F7F

Syntax **BITT** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution (Smem (15 – T(3–0))) → TC

Status Bits Affects TC

Description This instruction copies the specified bit of the data-memory value *Smem* into the TC bit in status register ST0. The four LSBs of T contain a bit code that specifies which bit is copied.

The bit address corresponds to (15 – T(3–0)). The bit code corresponds to the content of T(3–0).

Bit Codes for This Instruction

Bit Address	Bit Code	Bit Address	Bit Code
(LSB) 0	1111	8	0111
1	1110	9	0110
2	1101	10	0101
3	1100	11	0100
4	1011	12	0011
5	1010	13	0010
6	1001	14	0001
7	1000	(MSB) 15	0000

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
BITT *AR7+0
```

	Before Instruction		After Instruction
T	C	T	C
TC	0	TC	1
AR0	0008	AR0	0008
AR7	0100	AR7	0108
Data Memory			
0100h	0008	0100h	0008

Syntax **CALA**[**D**] *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

Nondelayed
(SP) – 1 → SP
(PC) + 1 → TOS
(src(15–0)) → PC

Delayed
(SP) – 1 → SP
(PC) + 3 → TOS
(src(15–0)) → PC

Status Bits None

Description This instruction passes control to the 16-bit address in the low part of *src* (bits 15–0). If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 6 cycles
4 cycles (delayed)

Classes Class 30B (see page 3-67)

Example 1

```
CALA A
```

	Before Instruction	After Instruction
A	00 0000 3000	00 0000 3000
PC	0025	3000
SP	1111	1110
Data Memory		
1110h	4567	0026

Example 2

```
CALAD B
ANDM 4444h, *AR1+
```

	Before Instruction		After Instruction
B	00 0000 2000	B	00 0000 2000
PC	0025	PC	2000
SP	1111	SP	1110
Data Memory			
1110h	4567	1110h	0028

After the memory location has been ANDed with 4444h, the program continues executing from location 2000h.

Syntax **CALL**[**D**] *pmad*

Operands 0 ≤ pmad ≤ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	Z	0	0	1	1	1	0	1	0	0
16-bit constant															

Execution

Nondelayed
(SP) − 1 → SP
(PC) + 2 → TOS
pmad → PC

Delayed
(SP) − 1 → SP
(PC) + 4 → TOS
pmad → PC

Status Bits None

Description This instruction passes control to the specified program-memory address (*pmad*). The return address is pushed onto the TOS before *pmad* is loaded into PC. If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable.

Words 2 words

Cycles 4 cycles
2 cycles (delayed)

Classes Class 29B (see page 3-66)

Example 1

```
CALL 3333h
```

	Before Instruction	After Instruction
PC	0025	3333
SP	1111	1110
Data Memory		
1110h	4567	0027

Example 2

```
CALLD 1000h
ANDM #4444h, *AR1+
```

	Before Instruction	After Instruction
PC	0025	1000
SP	1111	1110
Data Memory		
1110h	4567	0029

After the memory location has been ANDed with 4444h, the program continues executing from location 1000h.

Syntax **CC**[**D**] *pmad*, *cond* [, *cond* [, *cond*]]

Operands 0 ≤ pmad ≤ 65 535

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{BIO}$ low	0000 0011	NBIO	$\overline{BIO}$ high	0000 0010
C	C = 1	0000 1100	NC	C = 0	0000 1000
TC	TC = 1	0011 0000	NTC	TC = 0	0010 0000
AEQ	(A) = 0	0100 0101	BEQ	(B) = 0	0100 1101
ANEQ	(A) ≠ 0	0100 0100	BNEQ	(B) ≠ 0	0100 1100
AGT	(A) > 0	0100 0110	BGT	(B) > 0	0100 1110
AGEQ	(A) ≥ 0	0100 0010	BGEQ	(B) ≥ 0	0100 1010
ALT	(A) < 0	0100 0011	BLT	(B) < 0	0100 1011
ALEQ	(A) ≤ 0	0100 0111	BLEQ	(B) ≤ 0	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	Z	1	C	C	C	C	C	C	C	C
16-bit constant															

Execution

Nondelayed
If (cond(s))
Then
(SP) − 1 → SP
(PC) + 2 → TOS
pmad → PC
Else
(PC) + 2 → PC

Delayed
If (cond(s))
Then
    (SP) − 1 → SP
    (PC) + 4 → TOS
    pmad → PC
Else
    (PC) + 2 → PC

Status Bits Affects OVA or OVB (if OV or NOV is chosen)

Description This instruction passes control to the program-memory address (*pmad*) if the specified condition(s) is met. The two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory. If the condition(s) is met, the two words following the instruction are flushed from the pipeline and execution begins at *pmad*. If the condition(s) is not met, the PC is incremented by 2 and the two words following the instruction are executed.

If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction is fetched from program memory and executed. The two words following the delayed instruction have no effect on the conditions being tested. If the condition(s) is met, execution continues at *pmad*. If the condition(s) is not met, the PC is incremented by 2 and the two words following the delayed instruction are executed.

This instruction tests multiple conditions before passing control to another section of the program. This instruction can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

Group1: You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time. The accumulator must be the same for both conditions; you cannot test conditions for both accumulators with the same instruction. For example, you can test AGT and AOV at the same time, but you cannot test AGT and BOV at the same time.

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

Conditions for This Instruction

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

Note:

This instruction is not repeatable.

Words 2 words

Cycles 5 cycles (true condition)
3 cycles (false condition)
3 cycles (delayed)

Classes Class 31B (see page 3-69)

Example 1

```
CC 2222h, AGT
```

	Before Instruction		After Instruction
A	00 0000 3000	A	00 0000 3000
PC	0025	PC	2222
SP	1111	SP	1110
Data Memory			
1110h	4567	1110h	0027

Example 2

```
CCD 1000h, BOV
ANDM 4444h, *AR1+
```

	Before Instruction		After Instruction
PC	0025	PC	1000
OVB	1	OVB	0
SP	1111	SP	1110
Data Memory			
1110h	4567	1110h	0029

After the memory location has been ANDed with 4444h, the program continues executing from location 1000h.

Syntax **CMPL** *src* [, *dst*]

Operands src, dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution $(\overline{src}) \rightarrow dst$

Status Bits None

Description This instruction calculates the 1s complement of the content of *src* (this is a logical inversion). The result is stored in *dst*, if specified, or *src* otherwise.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
CMPL A, B
```

	Before Instruction		After Instruction
A	FC DFFA AEAA	A	FC DFFA AEAA
B	00 0000 7899	B	03 2005 5155

Syntax **CMPM** *Smem*, *#lk*

Operands Smem: Single data-memory operand
–32 768 ≤ lk ≤ 32 767

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution If (Smem) = lk
Then
  1 → TC
Else
  0 → TC

Status Bits Affects TC

Description This instruction compares the 16-bit single data-memory operand *Smem* to the 16-bit constant *lk*. If they are equal, TC is set to 1. Otherwise, TC is cleared to 0.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 6A (see page 3-10)
Class 6B (see page 3-11)

Example

```
CMPM *AR4+, 0404h
```

	Before Instruction		After Instruction
TC	1	TC	0
AR4	0100	AR4	0101
Data Memory			
0100h	4444	0100h	4444

Syntax **CMPR** *CC*, *ARx*

Operands $0 \leq CC \leq 3$
ARx: AR0–AR7

Opcode

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

Execution If (cond)
Then
  $1 \rightarrow TC$
Else
  $0 \rightarrow TC$

Status Bits Affects TC

Description This instruction compares the content of the designated auxiliary register (*ARx*) to the content of AR0 and sets the TC bit according to the comparison. The comparison is specified by the *CC* (condition code) value (see the following table). If the condition is true, TC is set to 1. If the condition is false, TC is cleared to 0. All conditions are computed as unsigned operations.

Condition	Condition Code (CC)	Description
EQ	00	Test if (ARx) = (AR0)
LT	01	Test if (ARx) < (AR0)
GT	10	Test if (ARx) > (AR0)
NEQ	11	Test if (ARx) ≠ (AR0)

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
CMPR 2, AR4
```

	Before Instruction	After Instruction
TC	1	0
AR0	FFFF	FFFF
AR4	7FFF	7FFF

Syntax **CMPS** *src*, *Smem*

Operands
src: A (accumulator A)
B (accumulator B)
Smem: Single data-memory operand

Opcode

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

Execution
If ((src(31–16)) > (src(15–0)))
Then
(src(31–16)) → Smem
(TRN) << 1 → TRN
0 → TRN(0)
0 → TC
Else
(src(15–0)) → Smem
(TRN) << 1 → TRN
1 → TRN(0)
1 → TC

Status Bits Affects TC

Description This instruction compares the two 16-bit 2s-complement values located in the high and low parts of *src* and stores the maximum value in the single data-memory location *Smem*. If the high part of *src* (bits 31–16) is greater, a 0 is shifted into the LSB of the transition register (TRN) and the TC bit is cleared to 0. If the low part of *src* (bits 15–0) is greater, a 1 is shifted into the LSB of TRN and the TC bit is set to 1.

This instruction does not follow the standard pipeline operation. The comparison is performed in the read phase; thus, the *src* value is the value one cycle before the instruction executes. TRN and the TC bit are updated during the execution phase.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 10A (see page 3-22)
Class 10B (see page 3-23)

Example

```
CMPS A, *AR4+
```

	Before Instruction		After Instruction
A	00 2345 7899	A	00 2345 7899
TC	0	TC	1
AR4	0100	AR4	0101
TRN	4444	TRN	8889
Data Memory			
0100h	0000	0100h	7899

Syntax **DADD** *Lmem, src* [*, dst*]

Operands
Lmem: Long data-memory operand
src, dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
If C16 = 0
Then
(Lmem) + (src) → dst
Else
(Lmem(31–16)) + (src(31–16)) → dst(39–16)
(Lmem(15–0)) + (src(15–0)) → dst(15–0)

Status Bits
Affected by SXM and OVM (only if C16 = 0)
Affects C and OVdst (or OVsrc, if dst is not specified)

Description This instruction adds the content of *src* to the 32-bit long data-memory operand *Lmem*. If a dst is specified, this instruction stores the result in *dst*. If no *dst* is specified, this instruction stores the result in *src*. The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. The 40-bit *src* value is added to the *Lmem*. The saturation and overflow bits are set according to the result of the operation.
- If C16 = 1, the instruction is executed in dual 16-bit mode. The high part of *src* (bits 31–16) is added to the 16 MSBs of *Lmem*, and the low part of *src* (bits 15–0) is added to the 16 LSBs of *Lmem*. The saturation and overflow bits are not affected in this mode. In this mode, the results are not saturated regardless of the state of the OVM bit.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes
Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example 1

```
DADD *AR3+, A, B
```

	Before Instruction		After Instruction
A	00 5678 8933	A	00 5678 8933
B	00 0000 0000	B	00 6BAC BD89
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Example 2

```
DADD *AR3−, A, B
```

	Before Instruction		After Instruction
A	00 5678 3933	A	00 5678 3933
B	00 0000 0000	B	00 6BAC 6D89
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Example 3

```
DADD *AR3−, A, B
```

	Before Instruction		After Instruction
A	00 5678 3933	A	00 5678 3933
B	00 0000 0000	B	00 8ACE 4E67
C16	0	C16	0
AR3	0101	AR3†	00FF
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax **DADST** *Lmem*, *dst*

Operands
Lmem: Long data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
If C16 = 1
Then
(Lmem(31–16)) + (T) → dst(39–16)
(Lmem(15–0)) – (T) → dst(15–0)
Else
(Lmem) + ((T) + (T) << 16) → dst

Status Bits
Affected by SXM and OVM (only if C16 = 0)
Affects C and OVdst

Description
This instruction adds the content of T to the 32-bit long data-memory operand *Lmem*. The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. *Lmem* is added to a 32-bit value composed of the content of T concatenated with the content of T left-shifted 16 bits (T <<16 + T). The result is stored in *dst*.
- If C16 = 1, the instruction is executed in dual 16-bit mode. The 16 MSBs of the *Lmem* are added to the content of T and stored in the upper 24 bits of *dst*. At the same time, the content of T is subtracted from the 16 LSBs of *Lmem*. The result is stored in the lower 16 bits of *dst*. In this mode, the results are not saturated regardless of the state of the OVM bit.

Note:

This instruction is meaningful only if C16 is set to 1 (dual 16-bit mode).

Words
1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles
1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes
Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example 1

```
DADST *AR3–, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	00 3879 1111
T	2345	T	2345
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Example 2

```
DADST *AR3+, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	00 3879 579B
T	2345	T	2345
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Syntax **DELAY** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution (Smem) → Smem + 1

Status Bits None

Description This instruction copies the content of a single data-memory location *Smem* into the next higher address. When data is copied, the content of the addressed location remains the same. This function is useful for implementing a Z delay in digital signal processing applications. The delay operation is also contained in the load T and insert delay (LTD) instruction (page 4-81) and the multiply by program memory and accumulate with delay (MACD) instruction (page 4-87).

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 24A (see page 3-56)
Class 24B (see page 3-56)

Example

```
DELAY *AR3
```

	Before Instruction		After Instruction
AR3	0100	AR3	0100
Data Memory			
0100h	6CAC	0100h	6CAC
0101h	0000	0101h	6CAC

Syntax **DLD** *Lmem*, *dst*

Operands

Lmem:	Long data-memory operand
dst:	A (accumulator A)
	B (accumulator B)

Opcode

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

Execution

If C16 = 0
Then
  (Lmem) → dst
Else
  (Lmem(31–16)) → dst(39–16)
  (Lmem(15–0)) → dst(15–0)

Status Bits Affected by SXM

Description This instruction loads *dst* with a 32-bit long operand *Lmem.* The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. *Lmem* is loaded to *dst.*
- If C16 = 1, the instruction is executed in dual 16-bit mode. The 16 MSBs of *Lmem* are loaded to the upper 24 bits of *dst*. At the same time, the 16 LSBs of *Lmem* are loaded in the lower 16 bits of *dst*.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example

```
DLD *AR3+, B
```

	Before Instruction		After Instruction
B	00 0000 0000	B	00 6CAC BD90
AR3	0100	AR3†	0102
Data Memory			
0100h	6CAC	0100h	6CAC
0101h	BD90	0101h	BD90

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Syntax **DRSUB** *Lmem*, *src*

Operands
Lmem: Long data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
If C16 = 0
Then
(Lmem) − (src) → src
Else
(Lmem(31–16)) − (src(31–16)) → src(39–16)
(Lmem(15–0)) − (src(15–0)) → src(15–0)

Status Bits
Affected by SXM and OVM (only if C16 = 0)
Affects C and OVsrc

Description
This instruction subtracts the content of *src* from the 32-bit long data-memory operand *Lmem* and stores the result in *src*. The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. The content of *src* (32 bits) is subtracted from *Lmem*. The result is stored in *src*.
- If C16 = 1, the instruction is executed in dual 16-bit mode. The high part of *src* (bits 31–16) is subtracted from the 16 MSBs of *Lmem* and the result is stored in the high part of *src* (bits 39–16). At the same time, the low part of *src* (bits 15–0) is subtracted from the 16 LSBs of *Lmem*. The result is stored in the low part of *src* (bits 15–0). In this mode, the results are not saturated regardless of the state of the OVM bit.

Words
1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles
1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes
Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example 1

```
DRSUB *AR3+, A
```

	Before Instruction		After Instruction
A	00 5678 8933	A	FF BEBB AB23
C	x	C	0
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Example 2

```
DRSUB *AR3–, A
```

	Before Instruction		After Instruction
A	00 5678 3933	A	FF BEBC FB23
C	1	C	0
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax **DSADT** *Lmem*, *dst*

Operands

Lmem: Long data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

If C16 = 1
Then
(Lmem(31–16)) − (T) → dst(39–16)
(Lmem(15–0)) + (T) → dst(15–0)
Else
(Lmem) – ((T) + (T << 16)) → dst

Status Bits

Affected by SXM and OVM (only if C16 = 0)
Affects C and OVdst

Description

This instruction subtracts/adds the content of T from the 32-bit long data-memory operand *Lmem* and stores the result in *dst*. The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. A 32-bit value composed of the content of T concatenated with the content of T left-shifted 16 bits (T << 16 + T) is subtracted from *Lmem*. The result is stored in *dst*.
- If C16 = 1, the instruction is executed in dual 16-bit mode. The content of T is subtracted from the 16 MSBs of *Lmem* and the result is stored in the high part of *dst* (bits 39–16). At the same time, the content of T is added to the 16 LSBs of *Lmem* and the result is stored in the low part of *dst* (bits 15–0). In this mode, the results are not saturated regardless of the state of the OVM bit.

Note:

This instruction is meaningful only if C16 is set (dual 16-bit mode).

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes

Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example 1

```
DSADT *AR3+, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	FF F1EF 1111
T	2345	T	2345
C	0	C	0
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Example 2

```
DSADT *AR3–, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	FF F1EF 579B
T	2345	T	2345
C	0	C	1
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax **DST** *src*, *Lmem*

Operands src: A (accumulator A)
B (accumulator B)
Lmem: Long data-memory operand

Opcode

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

Execution (src(31–0)) → Lmem

Status Bits None

Description This instruction stores the content of *src* in a 32-bit long data-memory location *Lmem*.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes Class 13A (see page 3-28)
Class 13B (see page 3-29)

Example 1

```
DST B, *AR3+
```

	Before Instruction		After Instruction
B	00 6CAC BD90	B	00 6CAC BD90
AR3	0100	AR3†	0102
Data Memory			
0100h	0000	0100h	6CAC
0101h	0000	0101h	BD90

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Example 2

```
DST B, *AR3–
```

	Before Instruction		After Instruction
B	00 6CAC BD90	B	00 6CAC BD90
AR3	0101	AR3†	00FF
Data Memory			
0100h	0000	0100h	BD90
0101h	0000	0101h	6CAC

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax **DSUB** *Lmem*, *src*

Operands

Lmem: Long data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

If C16 = 0
Then
(src) − (Lmem) → src
Else
(src(31–16)) − (Lmem(31–16)) → src(39–16)
(src(15–0)) − (Lmem(15–0)) → src(15–0)

Status Bits

Affected by SXM and OVM (only if C16 = 0)
Affects C and OVsrc

Description

This instruction subtracts the 32-bit long data-memory operand *Lmem* from the content of *src*, and stores the result in *src*. The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. *Lmem* is subtracted from the content of *src*.
- If C16 = 1, the instruction is executed in dual 16-bit mode. The 16 MSBs of *Lmem* are subtracted from the high part of *src* (bits 31–16) and the result is stored in the high part of *src* (bits 39–16). At the same time, the 16 LSBs of *Lmem* are subtracted from the low part of *src* (bits15–0) and the result is stored in the low part of *src* (bits 15–0).

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes

Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example 1

```
DSUB *AR3+, A
```

	Before Instruction		After Instruction
A	00 5678 8933	A	00 4144 54DD
C16	0	C16	0
AR3	0100	AR3†	0102

Data Memory

0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Example 2

```
DSUB *AR3–, A
```

	Before Instruction		After Instruction
A	00 5678 3933	A	00 4144 04DD
C	1	C	1
C16	1	C16	1
AR3	0100	AR3†	00FE

Data Memory

0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax **DSUBT** *Lmem*, *dst*

Operands
Lmem: Long data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
If C16 = 1
Then
(Lmem(31–16)) − (T) → dst(39–16)
(Lmem(15–0)) − (T) → dst(15–0)
Else
(Lmem) − ((T) + (T << 16)) → dst

Status Bits
Affected by SXM and OVM (only if C16 = 0)
Affects C and OVdst

Description
This instruction subtracts the content of T from the 32-bit long data-memory operand *Lmem* and stores the result in *dst*. The value of C16 determines the mode of the instruction:

- If C16 = 0, the instruction is executed in double-precision mode. A 32-bit value composed of the content of T concatenated with the content of T left-shifted 16 bits (T << 16 + T) is subtracted from *Lmem*. The result is stored in *dst.*
- If C16 = 1, the instruction is executed in dual 16-bit mode. The content of T is subtracted from the 16 MSBs of *Lmem* and the result is stored in the high part of *dst* (bits 39–16). At the same time, the content of T is subtracted from the 16 LSBs of *Lmem* and the result is stored in the low part of *dst* (bits 15–0). In this mode, the results are not saturated regardless of the value of the OVM bit.

Note:
This instruction is meaningful only if C16 is set to 1 (dual 16-bit mode).

Words
1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Lmem.

Cycles
1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Lmem.

Classes Class 9A (see page 3-20)
Class 9B (see page 3-21)

Example 1

```
DSUBT *AR3+, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	FF F1EF 1111
T	2345	T	2345
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Example 2

```
DSUBT *AR3–, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	FF F1EF 1111
T	2345	T	2345
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long operand instruction, AR3 is decremented by 2 after the execution.

Syntax **EXP** *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
If (src) = 0
Then
  0 → T
Else
  (Number of leading bits of src) − 8 → T

Status Bits None

Description This instruction computes the exponent value, which is a signed 2s-complement value in the –8 to 31 range, and stores the result in T. The exponent is computed by calculating the number of leading bits in *src* and subtracting 8 from this value. The number of leading bits is equivalent to the number of left shifts needed to eliminate the significant bits from the 40-bit *src* with the exception of the sign bit. The *src* is not modified after this instruction.

The result of subtracting 8 from the number of leading bits produces a negative exponent for accumulator values that have significant bits in the guard bits (the eight MSBs of the accumulator used in error detection and correction). See the normalization instruction (page 4-122).

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
EXP A
```

	Before Instruction			After Instruction	
A	FF FFFF FFCB	–53	A	FF FFFF FFCB	–53
T	0000		T	0019	25

Example 2

```
EXP B
```

	Before Instruction		After Instruction	
B	07 8543 2105	B	07 8543 2105	
T	FFFC	T	FFFC	–4†

† The value in accumulator B has significant bits in the guard bits, which results in a negative exponent.

Syntax **FB**[**D**] *extpmad*

Operands 0 ≤ extpmad ≤ 7F FFFF

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	Z	0	1	7-bit constant = pmad(22–16)						
16-bit constant = pmad(15–0)															

Execution (pmad(15–0)) → PC
(pmad(22–16)) → XPC

Status Bits None

Description This instruction passes control to the program-memory address *pmad* (bits 15–0) on the page specified by *pmad* (bits 22–16). The *pmad* can be either a symbolic or numeric address. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable. This instruction cannot be included in a block repeat (RPTB) instruction.

Words 2 words

Cycles 4 cycles
2 cycles (delayed)

Classes Class 29A (see page 3-66)

Example 1

```
FB 012000h
```

	Before Instruction	After Instruction
PC	1000	2000
XPC	00	01

2000h is loaded into the PC, 01h is loaded into XPC, and the program continues executing from that location.

Example 2

```
FBD 7F1000h
ANDM #4444h, *AR1+
```

	Before Instruction	After Instruction
PC	2000	1000
XPC	00	7F

After the operand has been ANDed with 4444h, the program continues executing from location 1000h on page 7Fh.

Syntax **FBACC**[**D**] *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (src(15–0)) → PC
(src(22–16)) → XPC

Status Bits None

Description This instruction loads the XPC with the value in *src* (bits 22–16) and passes control to the 16-bit address in the low part of *src* (bits 15–0). If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable. This instruction cannot be included in a block repeat (RPTB) instruction.

Words 1 word

Cycles 6 cycles
4 cycles (delayed)

Classes Class 30A (see page 3-67)

Example 1

```
FBACC A
```

	Before Instruction		After Instruction
A	00 0001 3000	A	00 0001 3000
PC	1000	PC	3000
XPC	00	XPC	01

1h is loaded into the XPC, 3000h is loaded into the PC, and the program continues executing from that location on page 1h.

Example 2

```
FBACCD B
ANDM 4444h *AR1+
```

	Before Instruction		After Instruction
B	00 007F 2000	B	00 007F 2000
XPC	01	XPC	7F

After the operand has been ANDed with 4444h value, 7Fh is loaded into the XPC, and the program continues executing from location 2000h on page 7Fh.

Syntax **FCALA**[**D**] *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

Nondelayed
(SP) – 1 → SP
(PC) + 1 → TOS
(SP) – 1 → SP
(XPC) → TOS
(src(15–0)) → PC
(src(22–16)) → XPC

Delayed
(SP) – 1 → SP
(PC) + 3 → TOS
(SP) – 1 → SP
(XPC) → TOS
(src(15–0)) → PC
(src(22–16)) → XPC

Status Bits None

Description This instruction loads the XPC with the value in *src* (bits 22–16) and passes control to the 16-bit address in the low part of *src* (bits 15–0). If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable. This instruction cannot be included in a block repeat (RPTB) instruction.

Words 1 word

Cycles 6 cycles
4 cycles (delayed)

Classes Class 30B (see page 3-67)

Example 1

```
FCALA A
```

	Before Instruction		After Instruction
A	00 007F 3000	A	00 007F 3000
PC	0025	PC	3000
XPC	00	XPC	7F
SP	1111	SP	110F
Data Memory			
1110h	4567	1110h	0026
110Fh	4567	110Fh	0000

Example 2

```
FCALAD B
ANDM #4444h, *AR1+
```

	Before Instruction		After Instruction
B	00 0020 2000	B	00 0020 2000
PC	0025	PC	2000
XPC	7F	XPC	20
SP	1111	SP	110F
Data Memory			
1110h	4567	1110h	0028
110Fh	4567	110Fh	007F

After the memory location has been ANDed with 4444h, the program continues executing from location 2000h on page 20h.

Syntax **FCALL**[**D**] *extpmad*

Operands 0 ≤ extpmad ≤ 7F FFFF

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	Z	1	1	7-bit constant = pmad(22–16)						
16-bit constant = pmad(15-0)															

Execution

Nondelayed
(SP) − 1 → SP
(PC) + 2 → TOS
(SP) – 1 → SP
(XPC) → TOS
(pmad(15–0)) → PC
(pmad(22–16)) → XPC

Delayed
(SP) – 1 → SP
(PC) + 4 → TOS
(SP) – 1 → SP
(XPC) → TOS
(pmad(15–0)) → PC
(pmad(22–16)) → XPC

Status Bits None

Description This instruction passes control to the specified program-memory address *pmad* (bits 15–0) on the page specified by *pmad* (bits 22–16). The return address is pushed onto the stack before *pmad* is loaded into PC. If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

Note:

This instruction is not repeatable. This instruction cannot be included in a block repeat (RPTB) instruction.

Words 2 words

Cycles 4 cycles
2 cycles (delayed)

Classes Class 29B (see page 3-66)

Example 1

```
FCALL 013333h
```

	Before Instruction		After Instruction
PC	0025	PC	3333
XPC	00	XPC	01
SP	1111	SP	110F
Data Memory			
1110h	4567	1110h	0027
110Fh	4567	110Fh	0000

Example 2

```
FCALLD 301000h
ANDM #4444h, *AR1+
```

	Before Instruction		After Instruction
PC	3001	PC	1000
XPC	7F	XPC	30
SP	1111	SP	110F
Data Memory			
1110h	4567	1110h	3005
110Fh	4567	110Fh	007F

After the memory location has been ANDed with 4444h, the program continues executing from location 1000h.

Syntax FIRS *Xmem*, *Ymem*, *pmad*

Operands Xmem, Ymem: Dual data-memory operands
$0 \le$ pmad $\le$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	0	X	X	X	X	Y	Y	Y	Y
16-bit constant															

Execution
pmad → PAR
While (RC) ≠ 0
(B) + (A(32–16)) × (Pmem addressed by PAR) → B
((Xmem) + (Ymem)) << 16 → A
(PAR) + 1 → PAR
(RC) − 1 → RC

Status Bits Affected by SXM, FRCT, and OVM
Affects C, OVA, and OVB

Description This instruction implements a symmetrical finite impulse respone (FIR) filter. This instruction multiplies accumulator A (bits 32–16) with a Pmem value addressed by *pmad* (in the program address register PAR) and adds the result to the value in accumulator B. At the same time, it adds the memory operands *Xmem* and *Ymem*, shifts the result left 16 bits, and loads this value into accumulator A. In the next iteration, *pmad* is incremented by 1. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Cycles 3 cycles

Classes Class 8 (see page 3-15)

Example

```
FIRS *AR3+, *AR4+, COEFFS
```

	Before Instruction	After Instruction
A	00 0077 0000	00 00FF 0000
B	00 0000 0000	00 0008 762C
FRCT	0	0
AR3	0100	0101
AR4	0200	0201
Data Memory		
0100h	0055	0055
0200h	00AA	00AA
Program Memory		
COEFFS	1234	1234

Syntax **FRAME** *K*

Operands $-128 \leq K \leq 127$

Opcode

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

Execution $(SP) + K \rightarrow SP$

Status Bits None

Description This instruction adds a short-immediate offset *K* to the SP. There is no latency for address generation in compiler mode (CPL = 1) or for stack manipulation by the instruction following this instruction.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example `FRAME 10h`

	Before Instruction		After Instruction
SP	1000	SP	1010

Syntax **FRET**[**D**]

Operands None

Opcode

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

Execution
(TOS) → XPC
(SP) + 1 → SP
(TOS) → PC
(SP) + 1 → SP

Status Bits None

Description This instruction replaces the XPC with the 7-bit value from the TOS and replaces the PC with the next 16-bit value on the stack. The SP is incremented by 1 for each of the two replacements. If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

Note:

This instruction is not repeatable.

Words 1 word

Cycles
6 cycles
4 cycles (delayed)

Classes Class 34 (see page 3-71)

Example FRET

	Before Instruction		After Instruction
PC	2112	PC	1000
XPC	01	XPC	05
SP	0300	SP	0302
Data Memory			
0300h	0005	0300h	0005
0301h	1000	0301h	1000

Syntax **FRETE**[**D**]

Operands None

Opcode

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

Execution
(TOS) → XPC
(SP) + 1 → SP
(TOS) → PC
(SP) + 1 → SP
0 → INTM

Status Bits Affects INTM

Description This instruction replaces the XPC with the 7-bit value from the TOS and replaces the PC with the next 16-bit value on the stack, continuing execution from the new PC value. This instruction automatically clears the interrupt mask bit (INTM) in ST1. (Clearing this bit enables interrupts.) If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 6 cycles
4 cycles (delayed)

Classes Class 34 (see page 3-71)

Example FRETE

	Before Instruction		After Instruction
PC	2112	PC	0110
XPC	05	XPC	6E
ST1	xCxx	ST1	x4xx
SP	0300	SP	0302
Data Memory			
0300h	006E	0300h	006E
0301h	0110	0301h	0110

Syntax **IDLE** *K*

Operands 1 ≤ K ≤ 3

Opcode

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

If K is:	NN is:
1	00
2	10
3	01

Execution (PC) +1 → PC

Status Bits Affected by INTM

Description This instruction forces the program being executed to wait until an unmasked interrupt or reset occurs. The PC is incremented by 1. The device remains in an idle state (power-down mode) until it is interrupted.

The idle state is exited after an unmasked interrupt, even if INTM = 1. If INTM = 1, the program continues executing at the instruction following the idle. If INTM = 0, the program branches to the corresponding interrupt service routine. The interrupt is enabled by the interrupt mask register (IMR), regardless of the INTM value. The following options, indicated by the value of K, determine the type of interrupts that can release the device from idle:

K = 1 Peripherals, such as the timer and the serial ports, are still active. The peripheral interrupts as well as reset and external interrupts release the processor from idle mode.

K = 2 Peripherals, such as the timer and the serial ports, are inactive. Reset and external interrupts release the processor from idle mode. Because interrupts are not latched in idle mode as they are in normal device operation, they must be low for a number of cycles to be acknowledged.

K = 3 Peripherals, such as the timer and the serial ports, are inactive and the PLL is halted. Reset and external interrupts release the processor from idle mode. Because interrupts are not latched in idle mode as they are in normal device operation, they must be low for a number of cycles to be acknowledged.

Note:

This instruction is not repeatable.

Words 1 word

Cycles The number of cycles needed to execute this instruction depends on the idle period. Because the entire device is halted when K = 3, the number of cycles cannot be specified. The minimum number of cycles is 4.

Classes Class 36 (see page 3-72)

Example 1

```
IDLE  1
```

The processor idles until a reset or unmasked interrupt occurs.

Example 2

```
IDLE  2
```

The processor idles until a reset or unmasked external interrupt occurs.

Example 3

```
IDLE  3
```

The processor idles until a reset or unmasked external interrupt occurs.

Syntax **INTR** *K*

Operands $0 \leq K \leq 31$

Opcode

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

Execution

(SP) − 1 → SP
(PC) + 1 → TOS
interrupt vector specified by K → PC
1 → INTM

Status Bits Affects INTM and IFR

Description This instruction transfers program control to the interrupt vector specified by *K*. This instruction allows you to use your application software to execute any interrupt service routine. For a list of interrupts and their corresponding *K* value, see your device datasheet.

During execution of the instruction, the PC is incremented by 1 and pushed onto the TOS. Then, the interrupt vector specified by *K* is loaded in the PC and the interrupt service routine for this interrupt is executed. The corresponding bit in the interrupt flag register (IFR) is cleared and interrupts are globally disabled (INTM = 1). The interrupt mask register (IMR) has no effect on the INTR instruction. INTR is executed regardless of the value of INTM.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 3 cycles

Classes Class 35 (see page 3-72)

Example

```
INTR 3
```

	Before Instruction		After Instruction
PC	0025	PC	FF8C
INTM	0	INTM	1
IPTR	01FF	IPTR	01FF
SP	1000	SP	0FFF
Data Memory			
0FFFh	9653	0FFFh	0026

Syntax

1: **LD** *Smem*, *dst*
2: **LD** *Smem*, **TS**, *dst*
3: **LD** *Smem*, **16**, *dst*
4: **LD** *Smem* [, *SHIFT*], *dst*
5: **LD** *Xmem*, *SHFT*, *dst*
6: **LD** *#K*, *dst*
7: **LD** *#lk* [, *SHFT*], *dst*
8: **LD** *#lk*, **16**, *dst*
9: **LD** *src*, **ASM** [, *dst*]
10: **LD** *src* [, *SHIFT*], *dst*

For additional load instructions, see *Load T/DP/ASM/ARP* on page 4-70.

Operands

Smem:	Single data-memory operand
Xmem:	Dual data-memory operand
src, dst:	A (accumulator A)
	B (accumulator B)

$0 \leq K \leq 255$
$-32\,768 \leq lk \leq 32\,767$
$-16 \leq SHIFT \leq 15$
$0 \leq SHFT \leq 15$

Opcode

1:

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

2:

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

3:

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

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	0	D	0	1	0	S	H	I	F	T

5:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	D	X	X	X	X	S	H	F	T

6:

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

7:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	0	1	0	S	H	F	T
16-bit constant															

8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	1	1	0	0	0	1	0
16-bit constant															

9:

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

10:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	1	0	S	H	I	F	T

Execution

1: (Smem) → dst
2: (Smem) << TS → dst
3: (Smem) << 16 → dst
4: (Smem) << SHIFT → dst
5: (Xmem) << SHFT → dst
6: K → dst
7: lk << SHFT → dst
8: lk << 16 → dst
9: (src) << ASM → dst
10: (src) << SHIFT → dst

Status Bits

Affected by SXM in all accumulator loads
Affected by OVM in loads with SHIFT or ASM shift
Affects OVdst (or OVsrc, when dst = src) in loads with SHIFT or ASM shift

Description

This instruction loads the accumulator (*dst*, or *src* if *dst* is not specified) with a data-memory value or an immediate value, supporting different shift quantities. Additionally, the instruction supports accumulator-to-accumulator moves with shift.

Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- Syntax 4: If *SHIFT* = 0, the instruction opcode is assembled as syntax 1.
- Syntax 4: If 0 < *SHIFT* ≤ 15 and Smem indirect addressing mode is included in Xmem, the instruction opcode is assembled as syntax 5.
- Syntax 5: If *SHFT* = 0, the instruction opcode is assembled as syntax 1.
- Syntax 7: If *SHFT* = 0 and 0 ≤ *lk* ≤ 255, the instruction opcode is assembled as syntax 6.

Words

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 word
Syntaxes 4, 7, and 8: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 cycle
Syntaxes 4, 7, and 8: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1, 2, 3, and 5: Class 3A (see page 3-5)
Syntaxes 1, 2, and 3: Class 3B (see page 3-6)
Syntax 4: Class 4A (see page 3-7)
Syntax 4: Class 4B (see page 3-8)
Syntaxes 6, 9, and 10: Class 1 (see page 3-3)
Syntaxes 7 and 8: Class 2 (see page 3-4)

Example 1

```
LD *AR1, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	00 0000 FEDC
SXM	0	SXM	0
AR1	0200	AR1	0200
Data Memory			
0200h	FEDC	0200h	FEDC

Example 2 `LD *AR1, A`

	Before Instruction		After Instruction
A	00 0000 0000	A	FF FFFF FEDC
SXM	1	SXM	1
AR1	0200	AR1	0200
Data Memory			
0200h	FEDC	0200h	FEDC

Example 3 `LD *AR1, TS, B`

	Before Instruction		After Instruction
B	00 0000 0000	B	FF FFFE DC00
SXM	1	SXM	1
AR1	0200	AR1	0200
T	8	T	8
Data Memory			
0200h	FEDC	0200h	FEDC

Example 4 `LD *AR3+, 16, A`

	Before Instruction		After Instruction
A	00 0000 0000	A	FF FEDC 0000
SXM	1	SXM	1
AR3	0300	AR1	0301
Data Memory			
0300h	FEDC	0300h	FEDC

Example 5 `LD #248, B`

	Before Instruction		After Instruction
B	00 0000 0000	B	00 0000 00F8
SXM	1	SXM	1

Example 6 `LD A, 8, B`

	Before Instruction		After Instruction
A	00 7FFD 0040	A	00 7FF0 0040
B	00 0000 FFFF	B	7F FD00 4000
OVB	0	OVB	1
SXM	1	SXM	1
Data Memory			
0200h	FEDC	0200h	FEDC

Syntax

1: **LD** *Smem*, **T**
2: **LD** *Smem*, **DP**
3: **LD** *#k9*, **DP**
4: **LD** *#k5*, **ASM**
5: **LD** *#k3*, **ARP**
6: **LD** *Smem*, **ASM**

For additional load instructions, see *Load Accumulator With Shift* on page 4-66.

Operands

Smem: Single data-memory operand
$0 \leq k9 \leq 511$
$-16 \leq k5 \leq 15$
$0 \leq k3 \leq 7$

Opcode

1:

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

2:

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

3:

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

4:

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

5:

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

6:

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

Execution

1: (Smem) → T
2: (Smem(8–0)) → DP
3: k9 → DP
4: k5 → ASM
5: k3 → ARP
6: (Smem(4–0)) → ASM

Status Bits

None

Description This instruction loads a value into T or into the DP, ASM, and ARP fields of ST0 or ST1. The value loaded can be a single data-memory operand *Smem* or a constant.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles Syntaxes 1, 3, 4, 5, and 6: 1 cycle
Syntax 2: 3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Syntaxes 1 and 6: Class 3A (see page 3-5)
Syntaxes 1 and 6: Class 3B (see page 3-6)
Syntax 2: Class 5A (see page 3-9)
Syntax 2: Class 5B (see page 3-9)
Syntaxes 3, 4, and 5: Class 1 (see page 3-3)

Example 1

```
LD *AR3+, T
```

	Before Instruction		After Instruction
T	0000	T	FEDC
AR3	0300	AR3	0301
Data Memory			
0300h	FEDC	0300h	FEDC

Example 2

```
LD *AR4, DP
```

	Before Instruction		After Instruction
AR4	0200	AR4	0200
DP	1FF	DP	0DC
Data Memory			
0200h	FEDC	0200h	FEDC

Example 3

```
LD #23, DP
```

	Before Instruction		After Instruction
DP	1FF	DP	017

Example 4

```
LD 15, ASM
```

	Before Instruction		After Instruction
ASM	00	ASM	0F

Example 5

```
LD 3, ARP
```

	Before Instruction		After Instruction
ARP	0	ARP	3

Example 6

```
LD 0, ASM
```

	Before Instruction		After Instruction
ASM	00	ASM	1C
DP	004	DP	004
Data Memory			
0200h	FEDC	0200h	FEDC

Syntax **LDM** *MMR, dst*

Operands

MMR: Memory-mapped register
dst: A (accumulator)
B (accumulator)

Opcode

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

Execution

(MMR) → dst(15–0)
00 0000h → dst(39–16)

Status Bits None

Description This instruction loads *dst* with the value in memory-mapped register *MMR*. The nine MSBs of the effective address are cleared to 0 to designate data page 0, regardless of the current value of DP or the upper nine bits of ARx. This instruction is not affected by the value of SXM.

Words 1 word

Cycles 1 cycle

Classes Class 3A (see page 3-5)

Example 1

```
LDM AR4, A
```

	Before Instruction		After Instruction
A	00 0000 1111	A	00 0000 FFFF
AR4	FFFF	AR4	FFFF

Example 2

```
LDM 060h, B
```

	Before Instruction		After Instruction
B	00 0000 0000	B	00 0000 1234
Data Memory			
0060h	1234	0060h	1234

Syntax

LD *Xmem*, *dst*
|| MAC[**R**] *Ymem* [, *dst_*]

Operands

dst: A (accumulator A)
B (accumulator B)
dst_: If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A
Xmem, Ymem: Dual data-memory operands

Opcode

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

Execution

(Xmem) << 16 → dst (31–16)
If (Rounding)
  Round (((Ymem) × (T)) + (dst_)) → dst_
Else
  ((Ymem) × (T)) + (dst_) → dst_

Status Bits

Affected by SXM, FRCT, and OVM
Affects OVdst_

Description

This instruction loads the high part of *dst* (bits 31–16) with a 16-bit dual data-memory operand *Xmem* shifted left 16-bits. In parallel, this instruction multiplies a dual data-memory operand *Ymem* by the content of T, adds the result of the multiplication to *dst_*, and stores the result in *dst_*.

If you use the R suffix, this instruction optionally rounds the result of the multiply and accumulate operation by adding 2^{15} to the result and clearing the LSBs (15–0) to 0, and stores the result in *dst_*.

Words

1 word

Cycles

1 cycle

Classes

Class 7 (see page 3-12)

Example 1

```
LD *AR4+, A
||MAC *AR5+, B
```

	Before Instruction	After Instruction
A	00 0000 1000	00 1234 0000
B	00 0000 1111	00 010C 9511
T	0400	0400
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
0100h	1234	1234
0200h	4321	4321

Example 2

```
LD *AR4+, A
||MACR *AR5+, B
```

	Before Instruction		After Instruction
A	00 0000 1000	A	00 1234 0000
B	00 0000 1111	B	00 010D 0000
T	0400	T	0400
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	1234	0100h	1234
0200h	4321	0200h	4321

Syntax

LD *Xmem*, *dst*
|| MAS[**R**] *Ymem* [, *dst_*]

Operands

Xmem, Ymem:	Dual data-memory operands
dst:	A (accumulator A)
	B (accumulator B)
dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A

Opcode

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

Execution

(Xmem) << 16 → dst (31–16)
If (Rounding)
  Round ((dst_) – ((T) × (Ymem))) → dst_
Else
  (dst_) – ((T) × (Ymem)) → dst_

Status Bits

Affected by SXM, FRCT, and OVM
Affects OVdst_

Description

This instruction loads the high part of *dst* (bits 31–16) with a 16-bit dual data-memory operand *Xmem* shifted left 16 bits. In parallel, this instruction multiplies a dual data-memory operand *Ymem* by the content of T, subtracts the result of the multiplication from *dst_*, and stores the result in *dst_*.

If you use the R suffix, this instruction optionally rounds the result of the multiply and subtract operation by adding 2^{15} to the result and clearing the LSBs (15–0) to 0, and stores the result in *dst_*.

Words

1 word

Cycles

1 cycle

Classes

Class 7 (see page 3-12)

Example 1

```
LD *AR4+, A
||MAS *AR5+, B
```

	Before Instruction		After Instruction
A	00 0000 1000	A	00 1234 0000
B	00 0000 1111	B	FF FEF3 8D11
T	0400	T	0400
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	1234	0100h	1234
0200h	4321	0200h	4321

Example 2

```
LD *AR4+, A
||MASR *AR5+, B
```

	Before Instruction		After Instruction
A	00 0000 1000	A	00 1234 0000
B	00 0000 1111	B	FF FEF4 0000
T	0400	T	0400
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	1234	0100h	1234
0200h	4321	0200h	4321

Syntax **LDR** *Smem*, *dst*

Operands

Smem: Single data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (Smem) << 16 + 1 << 15 → dst(31–16)

Status Bits Affected by SXM

Description This instruction loads the data-memory value *Smem* shifted left 16 bits into the high part of *dst* (bits 31–16). *Smem* is rounded by adding 2^{15} to this value and clearing the 15 LSBs (14–0) of the accumulator to 0. Bit 15 of the accumulator is set to 1.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
LDR *AR1, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	00 FEDC 8000
SXM	0	SXM	0
AR1	0200	AR1	0200
Data Memory			
0200h	FEDC	0200h	FEDC

Syntax **LDU** *Smem*, *dst*

Operands

Smem: Single data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

(Smem) → dst(15–0)
00 0000h → dst(39–16)

Status Bits None

Description This instruction loads the data-memory value *Smem* into the low part of *dst* (bits 15–0). The guard bits and the high part of *dst* (bits 39–16) are cleared to 0. Data is then treated as an unsigned 16-bit number. There is no sign extension regardless of the status of the SXM bit.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
LDU *AR1, A
```

	Before Instruction		After Instruction
A	00 0000 0000	A	00 0000 FEDC
AR1	0200	AR1	0200
Data Memory			
0200h	FEDC	0200h	FEDC

Syntax **LMS** *Xmem*, *Ymem*

Operands Xmem, Ymem: Dual data-memory operands

Opcode

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

Execution
$(A) + (Xmem) << 16 + 2^{15} \rightarrow A$
$(B) + (Xmem) \times (Ymem) \rightarrow B$

Status Bits
Affected by SXM, FRCT, and OVM
Affects C, OVA, and OVB

Description This instruction executes the least mean square (LMS) algorithm. The dual data-memory operand *Xmem* is shifted left 16 bits and added to accumulator A. The result is rounded by adding 2^{15} to the high part of the accumulator (bits 31–16). The result is stored in accumulator A. In parallel, *Xmem* and *Ymem* are multiplied and the result is added to accumulator B. *Xmem* does not overwrite T; therefore, T always contains the error value used to update coefficients.

Words 1 word

Cycles 1 cycle

Classes Class 7 (see page 3-12)

Example

```
LMS *AR3+, *AR4+
```

	Before Instruction	After Instruction
A	00 7777 8888	00 77CD 0888
B	00 0000 0100	00 0000 3972
FRCT	0	0
AR3	0100	0101
AR4	0200	0201
Data Memory		
0100h	0055	0055
0200h	00AA	00AA

Syntax **LTD** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution

(Smem) → T
(Smem) → Smem + 1

Status Bits None

Description This instruction copies the content of a single data-memory location *Smem* into T and into the address following this data-memory location. When data is copied, the content of the address location remains the same. This function is useful for implementing a Z delay in digital signal processing applications. This function also contains the memory delay instruction (page 4-41).

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 24A (see page 3-56)
Class 24B (see page 3-56)

Example

```
LTD *AR3
```

	Before Instruction	After Instruction
T	0000	6CAC
AR3	0100	0100
Data Memory		
0100h	6CAC	6CAC
0101h	xxxx	6CAC

Syntax

1: **MAC**[**R**] *Smem*, *src*
2: **MAC**[**R**] *Xmem*, *Ymem*, *src* [, *dst*]
3: **MAC** #*lk*, *src* [, *dst*]
4: **MAC** *Smem*, #*lk*, *src* [, *dst*]

Operands

Smem:	Single data-memory operands
Xmem, Ymem:	Dual data-memory operands
src, dst:	A (accumulator A)
	B (accumulator B)

$-32\,768 \leq \text{lk} \leq 32\,767$

Opcode

1:

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

2:

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

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	1	1	1
16-bit constant															

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	S	D	I	A	A	A	A	A	A	A
16-bit constant															

Execution

1: (Smem) × (T) + (src) → src
2: (Xmem) × (Ymem) + (src) → dst
(Xmem) → T
3: (T) × lk + (src) → dst
4: (Smem) × lk + (src) → dst
(Smem) → T

Status Bits

Affected by FRCT and OVM
Affects OVdst (or OVsrc, if dst is not specified)

Description

This instruction multiplies and adds with or without rounding. The result is stored in *dst* or *src*, as specified. For syntaxes 2 and 4, the data-memory value after the instruction is stored in T. T is updated in the read phase.

If you use the R suffix, this instruction rounds the result of the multiply and accumulate operation by adding 2^{15} to the result and clearing the LSBs (15–0) to 0.

Words

Syntaxes 1 and 2: 1 word
Syntaxes 3 and 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1 and 2: 1 cycle
Syntaxes 3 and 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntax 2: Class 7 (see page 3-12)
Syntax 3: Class 2 (see page 3-4)
Syntax 4: Class 6A (see page 3-10)
Syntax 4: Class 6B (see page 3-11)

Example 1

```
MAC *AR5+, A
```

	Before Instruction	After Instruction
A	00 0000 1000	00 0048 E000
T	0400	0400
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	1234	1234

Example 2

```
MAC #345h, A, B
```

	Before Instruction	After Instruction
A	00 0000 1000	00 0000 1000
B	00 0000 0000	00 001A 3800
T	0400	0400
FRCT	1	1

Example 3

```
MAC *AR5+, #1234h, A
```

	Before Instruction	After Instruction
A	00 0000 1000	00 0626 1060
T	0000	5678
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	5678	5678

Example 4

```
MAC *AR5+, *AR6+,A, B
```

	Before Instruction	After Instruction
A	00 0000 1000	00 0000 1000
B	00 0000 0004	00 0C4C 10C0
T	0008	5678
FRCT	1	1
AR5	0100	0101
AR6	0200	0201
Data Memory		
0100h	5678	5678
0200h	1234	1234

Example 5

```
MACR *AR5+, A
```

	Before Instruction	After Instruction
A	00 0000 1000	00 0049 0000
T	0400	0400
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	1234	1234

Example 6

```
MACR *AR5+, *AR6+,A, B
```

	Before Instruction	After Instruction
A	00 0000 1000	00 0000 1000
B	00 0000 0004	00 0C4C 0000
T	0008	5678
FRCT	1	1
AR5	0100	0101
AR6	0200	0201
Data Memory		
0100h	5678	5678
0200h	1234	1234

Syntax

1: **MACA**[**R**] *Smem* [, *B*]
2: **MACA**[**R**] **T**, *src* [, *dst*]

Operands

Smem: Single data-memory operand
src, dst: A (accumulator A)
B (accumulator B)

Opcode

1:

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

2:

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

Execution

1: $(Smem) \times (A(32–16)) + (B) \rightarrow B$
$(Smem) \rightarrow T$
2: $(T) \times (A(32–16)) + (src) \rightarrow dst$

Status Bits

Affected by FRCT and OVM
Affects OVdst (or OVsrc, if dst is not specified) and OVB in syntax 1

Description

This instruction multiplies the high part of accumulator A (bits 32–16) by a single data-memory operand *Smem* or by the content of T, adds the product to accumulator B (syntax 1) or to *src*. The result is stored in accumulator B (syntax 1) or in *dst* or *src* if no *dst* is specified. A(32–16) is used as a 17-bit operand for the multiplier.

If you use the R suffix, this instruction rounds the result of the multiply by accumulator A operation by adding 2^{15} to the result and clearing the 16 LSBs of *dst* (bits 15–0) to 0.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1 and 2: Class 3A (see page 3-5)
Syntaxes 1 and 2: Class 3B (see page 3-6)
Syntaxes 3 and 4: Class 1 (see page 3-3)

Example 1 `MACA *AR5+`

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0000 0000	B	00 0626 0060
T	0400	T	5678
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	5678	0100h	5678

Example 2 `MACA T, B, B`

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	00 009D 4BA0
T	0444	T	0444
FRCT	1	FRCT	1

Example 3 `MACAR *AR5+, B`

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0000 0000	B	00 0626 0000
T	0400	T	5678
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	5678	0100h	5678

Example 4 `MACAR T, B, B`

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	00 009D 0000
T	0444	T	0444
FRCT	1	FRCT	1

Syntax **MACD** *Smem*, *pmad*, *src*

Operands

Smem:	Single data-memory operand
src:	A (accumulator A)
	B (accumulator B)

$0 \le pmad \le 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	S	I	A	A	A	A	A	A	A
16-bit constant															

Execution

pmad → PAR
If (RC) ≠ 0
Then
    (Smem) × (Pmem addressed by PAR) + (src) → src
    (Smem) → T
    (Smem) → Smem + 1
    (PAR) + 1 → PAR
Else
    (Smem) × (Pmem addressed by PAR) + (src) → src
    (Smem) → T
    (Smem) → Smem + 1

Status Bits

Affected by FRCT and OVM
Affects OVsrc

Description

This instruction multiplies a single data-memory value *Smem* by a program-memory value *pmad*, adds the product to *src*, and stores the result in *src*. The data-memory value *Smem* is copied into T and into the next address following the *Smem* address. When this instruction is repeated, the program-memory address (in the program address register PAR) is incremented by 1. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction. This function also contains the memory delay instruction (page 4-41).

Words

2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 23A (see page 3-53)
Class 23B (see page 3-55)

Example

```
MACD *AR3–, COEFFS, A
```

	Before Instruction		After Instruction
A	00 0077 0000	A	00 007D 0B44
T	0008	T	0055
FRCT	0	FRCT	0
AR3	0100	AR3	00FF
Program Memory			
COEFFS	1234	COEFFS	1234
Data Memory			
0100h	0055	0100h	0055
0101h	0066	0101h	0055

Syntax **MACP** *Smem*, *pmad*, *src*

Operands

Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)
$0 \leq$ pmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	S	I	A	A	A	A	A	A	A
16-bit constant															

Execution

(pmad) → PAR
If (RC) ≠ 0
Then
(Smem) × (Pmem addressed by PAR) + (src) → src
(Smem) → T
(PAR) + 1 → PAR
Else
(Smem) × (Pmem addressed by PAR) + (src) → src
(Smem) → T

Status Bits

Affected by FRCT and OVM
Affects OVsrc

Description

This instruction multiplies a single data-memory value *Smem* by a program-memory value *pmad*, adds the product to *src*, and stores the result in *src*. The data-memory value *Smem* is copied into T. When this instruction is repeated, the program-memory address (in the program address register PAR) is incremented by 1. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words

2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 22A (see page 3-50)
Class 22B (see page 3-52)

Example

```
MACP *AR3–, COEFFS, A
```

	Before Instruction		After Instruction
A	00 0077 0000	A	00 007D 0B44
T	0008	T	0055
FRCT	0	FRCT	0
AR3	0100	AR3	00FF
Program Memory			
COEFFS	1234	COEFFS	1234
Data Memory			
0100h	0055	0100h	0055
0101h	0066	0101h	0066

Syntax **MACSU** *Xmem*, *Ymem*, *src*

Operands

Xmem, Ymem:	Dual data-memory operands
src:	A (accumulator A) B (accumulator B)

Opcode

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

Execution unsigned(Xmem) × signed(Ymem) + (src) → src
(Xmem) → T

Status Bits Affected by FRCT and OVM
Affects OVsrc

Description This instruction multiplies an unsigned data-memory value *Xmem* by a signed data-memory value *Ymem*, adds the product to *src*, and stores the result in *src*. The 16-bit unsigned value *Xmem* is stored in T. T is updated with the unsigned value *Xmem* in the read phase.

The data addressed by *Xmem* is fed from the D bus. The data addressed by *Ymem* is fed from the C bus.

Words 1 word

Cycles 1 cycle

Classes Class 7 (see page 3-12)

Example

```
MACSU *AR4+, *AR5+, A
```

	Before Instruction		After Instruction
A	00 0000 1000	A	00 09A0 AA84
T	0008	T	8765
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	8765	0100h	8765
0200h	1234	0200h	1234

Syntax **MAR** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution

In indirect addressing mode, the auxiliary register is modified as follows:
If compatibility is on (CMPT = 1), then:
  If (ARx = AR0)
    AR(ARP) is modified
    ARP is unchanged
  Else
    ARx is modified
    x → ARP
Else compatibility is off (CMPT = 0)
  ARx is modified
  ARP is unchanged

Status Bits

Affected by CMPT
Affects ARP (if CMPT = 1)

Description

This instruction modifies the content of the selected auxiliary register (ARx) as specified by *Smem*. In compatibility mode (CMPT = 1), this instruction modifies the ARx content as well as the auxiliary register pointer (ARP) value.

If CMPT = 0, the auxiliary register is modified but ARP is not.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 1 (see page 3-3)
Class 2 (see page 3-4)

Example 1

```
MAR *AR3+
```

	Before Instruction	After Instruction
CMPT	0	0
ARP	0	0
AR3	0100	0101

Example 2 `MAR *AR0–`

	Before Instruction		After Instruction
CMPT	1	CMPT	1
ARP	4	ARP	4
AR4	0100	AR4	00FF

Example 3 `MAR *AR3`

	Before Instruction		After Instruction
CMPT	1	CMPT	1
ARP	0	ARP	3
AR0	0008	AR0	0008
AR3	0100	AR3	0100

Example 4 `MAR *+AR3`

	Before Instruction		After Instruction
CMPT	1	CMPT	1
ARP	0	ARP	3
AR3	0100	AR3	0101

Example 5 `MAR *AR3–`

	Before Instruction		After Instruction
CMPT	1	CMPT	1
ARP	0	ARP	3
AR3	0100	AR3	00FF

Syntax

1: **MAS**[**R**] *Smem, src*
2: **MAS**[**R**] *Xmem*, *Ymem*, *src* [, *dst*]

Operands

Smem:	Single data-memory operand
Xmem, Ymem:	Dual data-memory operands
src, dst:	A (accumulator A)
	B (accumulator B)

Opcode

1:

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

2:

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

Execution

1: (src) – (Smem) $\times$ (T) $\rightarrow$ src
2: (src) – (Xmem) $\times$ (Ymem) $\rightarrow$ dst
(Xmem) $\rightarrow$ T

Status Bits

Affected by FRCT and OVM
Affects OVdst (or OVsrc, if dst = src)

Description

This instruction multiplies an operand by the content of T or multiplies two operands, subtracts the result from *src* unless *dst* is specified, and stores the result in *src* or *dst*. *Xmem* is loaded into T in the read phase.

If you use the R suffix, this instruction rounds the result of the multiply and subtract operation by adding 2^{15} to the result and clearing bits 15–0 of the result to 0.

The data addressed by *Xmem* is fed from DB and the data addressed by *Ymem* is fed from CB.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntax 2: Class 7 (see page 3-12)

Example 1 `MAS *AR5+, A`

	Before Instruction		After Instruction
A	00 0000 1000	A	FF FFB7 4000
T	0400	T	0400
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	1234	0100h	1234

Example 2 `MAS *AR5+, *AR6+, A, B`

	Before Instruction		After Instruction
A	00 0000 1000	A	00 0000 1000
B	00 0000 0004	B	FF F9DA 0FA0
T	0008	T	5678
FRCT	1	FRCT	1
AR5	0100	AR5	0101
AR6	0200	AR6	0201
Data Memory			
0100h	5678	0100h	5678
0200h	1234	0200h	1234

Example 3 `MASR *AR5+, A`

	Before Instruction		After Instruction
A	00 0000 1000	A	FF FFB7 0000
T	0400	T	0400
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	1234	0100h	1234

Example 4

```
MASR *AR5+, *AR6+, A, B
```

	Before Instruction		After Instruction
A	00 0000 1000	A	00 0000 1000
B	00 0000 0004	B	FF F9DA 0000
T	0008	T	5678
FRCT	1	FRCT	1
AR5	0100	AR5	0101
AR6	0200	AR6	0201
Data Memory			
0100h	5678	0100h	5678
0200h	1234	0200h	1234

Syntax

1: **MASA** *Smem* [, *B*]
2: **MASA**[**R**] **T**, *src* [, *dst*]

Operands

Smem: Single data-memory operand
src, dst: A (accumulator A)
B (accumulator B)

Opcode

1:

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

2:

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

Execution

1: (B) − (Smem) × (A(32–16)) → B
(Smem) → T
2: (src) − (T) × (A(32–16)) → dst

Status Bits

Affected by FRCT and OVM
Affects OVdst (or OVsrc, if dst is not specified) and OVB in syntax 1

Description

This instruction multiplies the high part of accumulator A (bits 32–16) by a single data-memory operand *Smem* or by the content of T, subtracts the result from accumulator B (syntax 1) or from *src*. The result is stored in accumulator B (syntax 1) or in *dst* or *src*, if no *dst* is specified. T is updated with the *Smem* value in the read phase.

If you use the R suffix in syntax 2, this instruction optionally rounds the result of the multiply by accumulator A and subtract operation by adding 2^{15} to the result and clearing bits 15–0 of the result to 0.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntax 2: Class 1 (see page 3-3)

Example 1

```
MASA *AR5+
```

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	FF F9DB FFA0
T	0400	T	5678
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	5678	0100h	5678

Example 2

```
MASA T, B
```

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	FF FF66 B460
T	0444	T	0444
FRCT	1	FRCT	1

Example 3

```
MASAR T, B
```

	Before Instruction		After Instruction
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	FF FF67 0000
T	0444	T	0444
FRCT	1	FRCT	1

Syntax **MAX** *dst*

Operands dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
If (A > B)
Then
  (A) → dst
  0 → C
Else
  (B) → dst
  1 → C

Status Bits Affects C

Description This instruction compares the content of the accumulators and stores the maximum value in *dst*. If the maximum value is in accumulator A, the carry bit, C, is cleared to 0; otherwise, it is set to 1.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
MAX A
```

	Before Instruction			After Instruction	
A	FFF6	–10	A	FFF6	–10
B	FFCB	–53	B	FFCB	–53
C	1		C	0	

Example 2

```
MAX A
```

	Before Instruction		After Instruction
A	00 0000 0055	A	00 0000 1234
B	00 0000 1234	B	00 0000 1234
C	0	C	1

Syntax **MIN** *dst*

Operands dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution If (A < B)
Then
(A) → dst
0 → C
Else
(B) → dst
1 → C

Status Bits Affects C

Description This instruction compares the content of the accumulators and stores the minimum value in *dst*. If the minimum value is in accumulator A, the carry bit, C, is cleared to 0; otherwise, it is set to 1.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
MIN A
```

	Before Instruction			After Instruction	
A	FFCB	–53	A	FFCB	–53
B	FFF6	–10	B	FFF6	–10
C	1		C	0	

Example 2

```
MIN A
```

	Before Instruction		After Instruction
A	00 0000 1234	A	00 0000 1234
B	00 0000 1234	B	00 0000 1234
C	0	C	1

Syntax

1: **MPY**[**R**] *Smem*, *dst*
2: **MPY** *Xmem*, *Ymem*, *dst*
3: **MPY** *Smem*, #*lk*, *dst*
4: **MPY** #*lk*, *dst*

Operands

Smem:	Single data-memory operand
Xmem, Ymem:	Dual data-memory operands
dst:	A (accumulator A)
	B (accumulator B)

$-32\,768 \leq \text{lk} \leq 32\,767$

Opcode

1:

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

2:

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

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	D	I	A	A	A	A	A	A	A
16-bit constant															

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	1	1	0	0	1	1	0
16-bit constant															

Execution

1: $(T) \times (Smem) \rightarrow dst$
2: $(Xmem) \times (Ymem) \rightarrow dst$
$(Xmem) \rightarrow T$
3: $(Smem) \times lk \rightarrow dst$
$(Smem) \rightarrow T$
4: $(T) \times lk \rightarrow dst$

Status Bits

Affected by FRCT and OVM
Affects OVdst

Description

This instruction multiplies the content of T or a data-memory value by a data-memory value or an immediate value, and stores the result in *dst*. T is loaded with the *Smem* or *Xmem* value in the read phase.

If you use the R suffix, this instruction optionally rounds the result of the multiply operation by adding 2^{15} to the result and then clearing bits 15–0 to 0.

Words

Syntaxes 1 and 2: 1 word
Syntaxes 3 and 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1 and 2: 1 cycle
Syntaxes 3 and 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntax 2: Class 7 (see page 3-12)
Syntax 3: Class 6A (see page 3-10)
Syntax 3: Class 6B (see page 3-11)
Syntax 4: Class 2 (see page 3-4)

Example 1

```
MPY 13, A
```

	Before Instruction	After Instruction
A	00 0000 0036	00 0000 0054
T	0006	0006
FRCT	1	1
DP	008	008
Data Memory		
040Dh	0007	0007

Example 2

```
MPY *AR2–, *AR4+0%, B;
```

	Before Instruction	After Instruction
B	FF FFFF FFE0	00 0000 0020
FRCT	0	0
AR0	0001	0001
AR2	01FF	01FE
AR4	0300	0301
Data Memory		
01FFh	0010	0010
0300h	0002	0002

Example 3

```
MPY #0FFFEh, A
```

	Before Instruction	After Instruction
A	000 0000 1234	FF FFFF C000
T	2000	2000
FRCT	0	0

Example 4

```
MPYR 0, B
```

	Before Instruction		After Instruction
B	FF FE00 0001	B	00 0626 0000
T	1234	T	1234
FRCT	0	FRCT	0
DP	004	DP	004
Data Memory			
0200h	5678	0200h	5678

Syntax

1: **MPYA** *Smem*
2: **MPYA** *dst*

Operands

Smem: Single data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

1:

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

2:

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

Execution

1: (Smem) $\times$ (A(32–16)) $\rightarrow$ B
(Smem) $\rightarrow$ T
2: (T) $\times$ (A(32–16)) $\rightarrow$ dst

Status Bits

Affected by FRCT and OVM
Affects OVdst (OVB in syntax 1)

Description

This instruction multiplies the high part of accumulator A (bits 32–16) by a single data-memory operand *Smem* or by the content of T, and stores the result in *dst* or accumulator B. T is updated in the read phase.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntax 2: Class 1 (see page 3-3)

Example 1

```
MPYA *AR2
```

	Before Instruction		After Instruction
A	FF 8765 1111	A	FF 8765 1111
B	00 0000 0320	B	FF D743 6558
T	1234	T	5678
FRCT	0	FRCT	0
AR2	0200	AR2	0200
Data Memory			
0200h	5678	0200h	5678

Example 2 `MPYA B`

	Before Instruction		After Instruction
A	FF 8765 1111	A	FF 8765 1111
B	00 0000 0320	B	FF DF4D B2A3
T	4567	T	4567
FRCT	0	FRCT	0

Syntax **MPYU** *Smem*, *dst*

Operands
Smem: Single data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution unsigned(T) $\times$ unsigned(Smem) $\rightarrow$ dst

Status Bits
Affected by FRCT and OVM
Affects OVdst

Description This instruction multiplies the unsigned content of T by the unsigned content of the single data-memory operand *Smem*, and stores the result in *dst.* The multiplier acts as a signed 17 $\times$ 17-bit multiplier for this instruction with the MSB of both operands cleared to 0. This instruction is particularly useful for computing multiple-precision products, such as multiplying two 32-bit numbers to yield a 64-bit product.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes
Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
MPYU *AR0–, A
```

	Before Instruction	After Instruction
A	FF 8000 0000	00 3F80 0000
T	4000	4000
FRCT	0	0
AR0	1000	0FFF
Data Memory		
1000h	FE00	FE00

Syntax **MVDD** *Xmem*, *Ymem*

Operands Xmem, Ymem: Dual data-memory operands

Opcode

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

Execution (Xmem) → Ymem

Status Bits None

Description This instruction copies the content of the data-memory location addressed by *Xmem* to the data-memory location addressed by *Ymem*.

Words 1 word

Cycles 1 cycle

Classes Class 14 (see page 3-30)

Example

```
MVDD *AR3+, *AR5+
```

	Before Instruction	After Instruction
AR3	8000	8001
AR5	0200	0201
Data Memory		
0200h	ABCD	1234
8000h	1234	1234

Syntax **MVDK** *Smem*, *dmad*

Operands Smem: Single data-memory operand
$0 \leq$ dmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution (dmad) → EAR
If (RC) ≠ 0
Then
  (Smem) → Dmem addressed by EAR
  (EAR) + 1 → EAR
Else
  (Smem) → Dmem addressed by EAR

Status Bits None

Description This instruction copies the content of a single data-memory operand *Smem* to a data-memory location addressed by a 16-bit immediate value *dmad* (address is in the EAB address register EAR). You can use this instruction with the single-repeat instruction to move consecutive words in data memory (using indirect addressing). The number of words to be moved is one greater than the number contained in the repeat counter at the beginning of the instruction. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 19A (see page 3-40)
Class 19B (see page 3-42)

Example 1

```
MVDK 10, 8000h
```

	Before Instruction	After Instruction
DP	004	004
Data Memory		
020Ah	1234	1234
8000h	ABCD	1234

Example 2

```
MVDK *AR3–, 1000h
```

	Before Instruction		After Instruction
AR3	01FF	AR3	01FE
Data Memory			
1000h	ABCD	1000h	1234
01FFh	1234	01FFh	1234

Syntax **MVDM** *dmad*, *MMR*

Operands MMR: Memory-mapped register
$0 \leq \text{dmad} \leq 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution
dmad → DAR
If (RC) ≠ 0
Then
  (Dmem addressed by DAR) → MMR
  (DAR) + 1 → DAR
Else
  (Dmem addressed by DAR) → MMR

Status Bits None

Description This instruction copies data from a data-memory location *dmad* (address is in the DAB address register DAR) to a memory-mapped register *MMR*. The data-memory value is addressed with a 16-bit immediate value. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Cycles 2 cycles

Classes Class 19A (see page 3-40)

Example

```
MVDM 300h, BK
```

	Before Instruction		After Instruction
BK	ABCD	BK	1234
Data Memory			
0300h	1234	0300h	1234

Syntax **MVDP** *Smem*, *pmad*

Operands
Smem: Single data-memory operand
$0 \leq$ pmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution
pmad → PAR
If (RC) ≠ 0
Then
  (Smem) → Pmem addressed by PAR
  (PAR) + 1 → PAR
Else
  (Smem) → Pmem addressed by PAR

Status Bits None

Description This instruction copies a 16-bit single data-memory operand *Smem* to a program-memory location addressed by a 16-bit immediate value *pmad*. You can use this instruction with the repeat instruction to move consecutive words in data memory (using indirect addressing) to the contiguous program-memory space addressed by 16-bit immediate values. The source and destination blocks do not have to be entirely on-chip or off-chip. When used with repeat, this instruction becomes a single-cycle instruction after the repeat pipeline starts. In addition, when repeat is used with this instruction, interrupts are inhibited. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 4 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 20A (see page 3-44)
Class 20B (see page 3-46)

Example

```
MVDP 0, 0FE00h
```

	Before Instruction		After Instruction
DP	004	DP	004
Data Memory			
0200h	0123	0200h	0123
Program Memory			
FE00h	FFFF	FE00h	0123

Syntax **MVKD** *dmad*, *Smem*

Operands Smem: Single data-memory operand
$0 \leq$ dmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution dmad → DAR
If (RC) ≠ 0
Then
(Dmem addressed by DAR) → Smem
(DAR) + 1 → DAR
Else
(Dmem addressed by DAR) → Smem

Status Bits None

Description This instruction moves data from data memory to data memory. The source data-memory value is addressed with a 16-bit immediate operand *dmad* and is moved to *Smem*. You can use this instruction with the single repeat instruction to move consecutive words in data memory (using indirect addressing). The number of words to move is one greater than the number contained in the repeat counter at the beginning of the instruction. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 19A (see page 3-40)
Class 19B (see page 3-42)

Example 1

```
MVKD 300h, 0
```

	Before Instruction	After Instruction
DP	004	004
Data Memory		
0200h	ABCD	1234
0300h	1234	1234

Example 2

```
MVKD 1000h, *+AR5
```

	Before Instruction		After Instruction
AR5	01FF	AR5	0200
Data Memory			
1000h	1234	1000h	1234
0200h	ABCD	0200h	1234

Syntax **MVMD** *MMR*, *dmad*

Operands MMR: Memory-mapped register
$0 \leq$ dmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution dmad → EAR
If (RC) ≠ 0
Then
  (MMR) → Dmem addressed by EAR
  (EAR) + 1 → EAR
Else
  (MMR) → Dmem addressed by EAR

Status Bits None

Description This instruction moves data from a memory-mapped register *MMR* to data memory. The data-memory destination is addressed with a 16-bit immediate value *dmad*. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Cycles 2 cycles

Classes Class 19A (see page 3-40)

Example

```
MVMD AR7, 8000h
```

	Before Instruction		After Instruction
AR7	1234	AR7	1234
Data Memory			
8000h	ABCD	8000h	1234

Syntax **MVMM** *MMRx*, *MMRy*

Operands
MMRx: AR0–AR7, SP
MMRy: AR0–AR7, SP

Opcode

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

Register	MMRX/MMRY	Register	MMRX/MMRY
AR0	0000	AR5	0101
AR1	0001	AR6	0110
AR2	0010	AR7	0111
AR3	0011	SP	1000
AR4	0100		

Execution (MMRx) → MMRy

Status Bits None

Description This instruction moves the content of memory-mapped register *MMRx* to the memory-mapped register *MMRy*. Only nine operands are allowed: AR0–AR7 and SP. The read operation from *MMRx* is executed in the decode phase. The write operation to *MMRy* is executed in the access phase.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
MVMM SP, AR1
```

	Before Instruction		After Instruction
AR1	3EFF	AR1	0200
SP	0200	SP	0200

Syntax **MVPD** *pmad*, *Smem*

Operands Smem: Single data-memory operand
$0 \leq$ pmad $\leq$ 65 535

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution
pmad → PAR
If (RC) ≠ 0
Then
    (Pmem addressed by PAR) → Smem
    (PAR) + 1 → PAR
Else
    (Pmem addressed by PAR) → Smem

Status Bits None

Description This instruction moves a word in program memory addressed by a 16-bit immediate value *pmad* to a data-memory location addressed by *Smem*. This instruction can be used with the repeat instruction to move consecutive words addressed by a 16-bit immediate program address to contiguous data-memory locations addressed by *Smem*. The source and destination blocks do not have to be entirely on-chip or off-chip. When used with repeat, this instruction becomes a single-cycle instruction after the repeat pipeline starts. In addition, when repeat is used with this instruction, interrupts are inhibited. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 21A (see page 3-47)
Class 21B (see page 3-49)

Example 1 `MVPD 0FE00h, 5`

	Before Instruction		After Instruction
DP	006	DP	006
Program Memory			
FE00h	8A55	FE00h	8A55
Data Memory			
0305h	FFFF	0305h	8A55

Example 2 `MVPD 2000h, *AR7–0`

	Before Instruction		After Instruction
AR0	0002	AR0	0002
AR7	0FFE	AR7	0FFC
Program Memory			
2000h	1234	2000h	1234
Data Memory			
0FFEh	ABCD	0FFEh	1234

Syntax **NEG** *src* [, *dst*]

Operands src, dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (src) × –1 → dst

Status Bits Affected by OVM
Affects C and OVdst (or OVsrc, when dst = src)

Description This instruction computes the 2s complement of the content of *src* (either A or B) and stores the result in *dst* or *src*, if *dst* is not specified. This instruction clears the carry bit, C, to 0 for all nonzero values of the accumulator. If the accumulator equals 0, the carry bit is set to 1.

If the accumulator equals FF 8000 0000h, the negate operation causes an overflow because the 2s complement of FF 8000 0000h exceeds the lower 32 bits of the accumulator. If OVM = 1, *dst* is assigned 00 7FFF FFFFh. If OVM = 0, *dst* is assigned 00 8000 0000h. The OV bit for *dst* is set to indicate overflow in either case.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
NEG A, B
```

	Before Instruction		After Instruction
A	FF FFFF F228	A	FF FFFF F228
B	00 0000 1234	B	00 0000 0DD8
OVA	0	OVA	0

Example 2

```
NEG B, A
```

	Before Instruction		After Instruction
A	00 0000 1234	A	FF 8000 0000
B	00 8000 0000	B	00 8000 0000
OVB	0	OVB	0

Example 3

```
NEG A
```

	Before Instruction		After Instruction
A	80 0000 0000	A	80 0000 0000
OVA	0	OVA	1
OVM	0	OVM	0

Example 4 `NEG A`

	Before Instruction		After Instruction
A	80 0000 0000	A	00 7FFF FFFF
OVA	0	OVA	1
OVM	1	OVM	1

Syntax **NOP**

Operands None

Opcode

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

Execution None

Status Bits None

Description No operation is performed. Only the PC is incremented. This is useful to create pipeline and execution delays.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
NOP
```

No operation is performed.

Syntax **NORM** *src* [, *dst*]

Operands src, dst : A (accumulator A)
B (accumulator B)

Opcode

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

Execution (src) << TS → dst

Status Bits Affected by SXM and OVM
Affects OVdst (or OVsrc, when dst = src)

Description The signed number contained in *src* is normalized and the value is stored in *dst* or *src*, if *dst* is not specified. Normalizing a fixed-point number separates the number into a mantissa and an exponent by finding the magnitude of the sign-extended number.

This instruction allows single-cycle normalization of the accumulator once the EXP instruction, which computes the exponent of a number, has executed. The shift value is defined by T(5–0) and coded as a 2s-complement number. The valid shift values are –16 to 31. For the normalization, the shifter needs the shift value (in T) in the read phase; the normalization is executed in the execution phase.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
NORM A
```

	Before Instruction		After Instruction
A	FF FFFF F001	A	FF 8008 0000
T	0013	T	0013

Example 2

```
NORM B, A
```

	Before Instruction		After Instruction
A	FF FFFF F001	A	00 4214 1414
B	21 0A0A 0A0A	B	21 0A0A 0A0A
T	0FF9	T	0FF9

Syntax

1: **OR** *Smem, src*
2: **OR** *#lk* [, *SHFT*], *src* [, *dst*]
3: **OR** *#lk,* **16**, *src* [, *dst*]
4: **OR** *src* [*, SHIFT*], [, *dst*]

Operands

src, dst : A (accumulator A)
B (accumulator B)
Smem : Single data-memory operand
$0 \leq$ SHFT ≤ 15
$-16 \leq$ SHIFT ≤ 15
$0 \leq$ lk $\leq$ 65 535

Opcode

1:

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

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	0	0	S	H	F	T
16-bit constant															

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	1	0	0
16-bit constant															

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	1	0	1	S	H	I	F	T

Execution

1: (Smem) OR (src(15–0)) → src
src(39–16) unchanged
2: lk << SHFT OR (src) → dst
3: lk << 16 OR (src) → dst
4: (src or [dst]) OR (src) << SHIFT → dst

Status Bits

None

Description

This instruction ORs the *src* with a single data-memory operand *Smem*, a left-shifted 16-bit immediate value *lk*, *dst*, or with itself. The result is stored in *dst,* or *src* if *dst* is not specified. The values can be shifted as indicated by the instruction. For a positive (left) shift, low-order bits are cleared and high-order bits are not sign extended. For a negative (right) shift, high-order bits are not sign extended.

Words

Syntaxes 1 and 4: 1 word
Syntaxes 2 and 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1 and 4: 1 cycle
Syntaxes 2 and 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntaxes 2 and 3: Class 2 (see page 3-4)
Syntax 4: Class 1 (see page 3-3)

Example 1

```
OR *AR3+, A
```

	Before Instruction		After Instruction
A	00 00FF 1200	A	00 00FF 1700
AR3	0100	AR3	0101
Data Memory			
0100h	1500	0100h	1500

Example 2

```
OR A, +3, B
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	00 0000 9800

Syntax **ORM** *#lk, Smem*

Operands Smem: Single data-memory operand
$0 \leq lk \leq 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution lk OR (Smem) → Smem

Status Bits None

Description This instruction ORs the single data-memory operand *Smem* with a 16-bit constant *lk*, and stores the result in *Smem.* This instruction is a memory-to-memory operation.

Note:

This instruction is not repeatable.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 18A (see page 3-39)
Class 18B (see page 3-39)

Example

```
ORM 0404h, *AR4+
```

	Before Instruction		After Instruction
AR4	0100	AR4	0101
Data Memory			
0100h	4444	0100h	4444

Syntax **POLY** *Smem*

Operands Smem : Single data-memory operand

Opcode

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

Execution

Round (A(32–16) × (T) + (B)) → A
(Smem) << 16 → B

Status Bits

Affected by FRCT, OVM, and SXM
Affects OVA

Description

This instruction shifts the content of the single data-memory operand *Smem* 16 bits to the left and stores the result in accumulator B. In parallel, this instruction multiplies the high part of accumulator A (bits 32–16) by the content of T, adds the product to accumulator B, rounds the result of this operation, and stores the final result in accumulator A. This instruction is useful for polynomial evaluation to implement computations that take one cycle per monomial to execute.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
POLY *AR3+%
```

	Before Instruction		After Instruction
A	00 1234 0000	A	00 0627 0000
B	00 0001 0000	B	00 2000 0000
T	5678	T	5678
AR3	0200	AR3	0201
Data Memory			
0200h	2000	0200h	2000

Syntax **POPD** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution (TOS) → Smem
(SP) + 1 → SP

Status Bits None

Description This instruction moves the content of the data-memory location addressed by SP to the memory location specified by *Smem*. SP is incremented by 1.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 17A (see page 3-36)
Class 17B (see page 3-38)

Example

```
POPD 10
```

	Before Instruction	After Instruction
DP	008	008
SP	0300	0301
Data Memory		
0300h	0092	0092
040Ah	0055	0092

Syntax **POPM** *MMR*

Operands MMR: Memory-mapped register

Opcode

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

Execution (TOS) → MMR
(SP) + 1 → SP

Status Bits None

Description This instruction moves the content of the data-memory location addressed by SP to the specified memory-mapped register *MMR*. SP is incremented by 1.

Words 1 word

Cycles 1 cycle

Classes Class 17A (see page 3-36)

Example

```
POPM AR5
```

	Before Instruction		After Instruction
AR5	0055	AR5	0060
SP	03F0	SP	03F1
Data Memory			
03F0h	0060	03F0h	0060

Syntax **PORTR** *PA*, *Smem*

Operands Smem: Single data-memory operand
$0 \leq PA \leq 65\,535$

Opcode

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

Execution (PA) → Smem

Status Bits None

Description This instruction reads a 16-bit value from an external I/O port *PA* (16-bit immediate address) into the specified data-memory location *Smem*. The $\overline{\text{IS}}$ signal goes low to indicate an I/O access, and the $\overline{\text{IOSTRB}}$ and READY timings are the same as for an external data memory read.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles (dependent on the external I/O operation)

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 27A (see page 3-63)
Class 27B (see page 3-63)

Example

```
PORTR 05, INDAT ;INDAT .equ 60h
```

	Before Instruction		After Instruction
DP	000	DP	000
I/O Memory			
0005h	7FFA	0005h	7FFA
Data Memory			
0060h	0000	0060h	7FFA

Syntax **PORTW** *Smem, PA*

Operands Smem: Single data-memory operand
$0 \leq PA \leq 65\,535$

Opcode

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

Execution (Smem) → PA

Status Bits None

Description This instruction writes a 16-bit value from the specified data-memory location *Smem* to an external I/O port *PA*. The $\overline{\text{IS}}$ signal goes low to indicate an I/O access, and the $\overline{\text{IOSTRB}}$ and READY timings are the same as for an external data memory read.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles (dependent on the external I/O operation)

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 28A (see page 3-64)
Class 28B (see page 3-65)

Example

```
PORTW OUTDAT, 5h ; OUTDAT .equ 07h
```

	Before Instruction		After Instruction
DP	001	DP	001
I/O Memory			
0005h	0000	0005h	7FFA
Data Memory			
0087h	7FFA	0087h	7FFA

Syntax **PSHD** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution
(SP) − 1 → SP
(Smem) → TOS

Status Bits None

Description After SP has been decremented by 1, this instruction stores the content of the memory location *Smem* in the data-memory location addressed by SP. SP is read during the decode phase; it is stored during the access phase.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes
Class 16A (see page 3-33)
Class 16B (see page 3-35)

Example

```
PSHD *AR3+
```

	Before Instruction		After Instruction
AR3	0200	AR3	0201
SP	8000	SP	7FFF
Data Memory			
0200h	07FF	0200h	07FF
7FFFh	0092	7FFFh	07FF

Syntax **PSHM** *MMR*

Operands MMR: Memory-mapped register

Opcode

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

Execution (SP) − 1 → SP
(MMR) → TOS

Status Bits None

Description After SP has been decremented by 1, this instruction stores the content of the memory-mapped register *MMR* in the data-memory location addressed by SP.

Words 1 word

Cycles 1 cycle

Classes Class 16A (see page 3-33)

Example

```
PSHM BRC
```

	Before Instruction		After Instruction
BRC	1234	BRC	1234
SP	2000	SP	1FFF
Data Memory			
1FFFh	07FF	1FFFh	1234

Syntax **RC**[**D**] *cond* [, *cond* [, *cond*]]

Operands The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{BIO}$ low	0000 0011	NBIO	$\overline{BIO}$ high	0000 0010
C	C = 1	0000 1100	NC	C = 0	0000 1000
TC	TC = 1	0011 0000	NTC	TC = 0	0010 0000
AEQ	(A) = 0	0100 0101	BEQ	(B) = 0	0100 1101
ANEQ	(A) $\neq$ 0	0100 0100	BNEQ	(B) $\neq$ 0	0100 1100
AGT	(A) $>$ 0	0100 0110	BGT	(B) $>$ 0	0100 1110
AGEQ	(A) $\geq$ 0	0100 0010	BGEQ	(B) $\geq$ 0	0100 1010
ALT	(A) $<$ 0	0100 0011	BLT	(B) $<$ 0	0100 1011
ALEQ	(A) $\leq$ 0	0100 0111	BLEQ	(B) $\leq$ 0	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Opcode

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

Execution If (cond(s))
Then
  (TOS) → PC
  (SP) + 1 → SP
Else
  (PC) + 1 → PC

Status Bits None

Description If the conditions given by *cond* are met, this instruction replaces the PC with the data-memory value from the TOS and increments the SP by 1. If the conditions are not met, this instruction just increments the PC by 1.

If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed. The two instruction words following this instruction have no effect on the condition(s) being tested.

This instruction tests multiple conditions before passing control to another section of the program. It can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

Group 1 You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time. The accumulator must be the same for both conditions; you cannot test conditions for both accumulators with the same instruction. For example, you can test AGT and AOV at the same time, but you cannot test AGT and BOV at the same time.

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

Conditions for This Instruction

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

Note:

This instruction is not repeatable.

Words 1 word

Cycles 5 cycles (true condition)
3 cycles (false condition)
3 cycles (delayed)

Classes Class 32 (see page 3-70)

Example

```
RC AGEQ, ANOV        ; return is executed if the accumulator A
                     ; contents are positive and the OVA bit
                     ; is a zero
```

	Before Instruction		After Instruction
PC	0807	PC	2002
OVA	0	OVA	0
SP	0308	SP	0309
Data Memory			
0308h	2002	0308h	2002

Syntax **READA** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution

A → PAR
If ((RC) ≠ 0)
    (Pmem (addressed by PAR)) → Smem
    (PAR) + 1 → PAR
    (RC) – 1 → RC
Else
    (Pmem (addressed by PAR)) → Smem

Status Bits None

Description This instruction transfers a word from a program-memory location specified by accumulator A to a data-memory location specifed by *Smem*. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction. The program-memory location is defined by Accumulator A, depending on the specific device, as follows:

C541–C546	Devices with Extended Program Memory
A(15–0)	A(22–0)

This instruction can be used with the repeat instruction to move consecutive words (starting with the address specified in accumulator A) to a contiguous data-memory space addressed using indirect addressing. Source and destination blocks do not need to be entirely on-chip or off-chip.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 5 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 25A (see page 3-57)
Class 25B (see page 3-59)

Example `READA 6`

	Before Instruction		After Instruction
A	00 0000 0023	A	00 0000 0023
DP	004	DP	004
Program Memory			
0023h	0306	0023h	0306
Data Memory			
0206h	0075	0206h	0306

Syntax **RESET**

Operands None

Opcode

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

Execution These fields of PMST, ST0, and ST1 are loaded with the values shown:

(IPTR) << 7 → PC	0 → OVA	0 → OVB
1 → C	1 → TC	0 → ARP
0 → DP	1 → SXM	0 → ASM
0 → BRAF	0 → HM	1 → XF
0 → C16	0 → FRCT	0 → CMPT
0 → CPL	1 → INTM	0 → IFR
0 → OVM		

Status Bits The status bits affected are listed in the execution section.

Description This instruction performs a nonmaskable software reset that can be used at any time to put the ’54x into a known state. When the reset instruction is executed, the operations listed in the execution section occur. The MP/$\overline{\text{MC}}$ pin is not sampled during this software reset. The initialization of IPTR and the peripheral registers is different from the initialization using $\overline{\text{RS}}$. This instruction is not affected by INTM; however, it sets INTM to 1 to disable interrupts.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 3 cycles

Classes Class 35 (see page 3-72)

Example `RESET`

	Before Instruction	After Instruction
PC	0025	0080
INTM	0	1
IPTR	1	1

Syntax **RET**[**D**]

Operands None

Opcode

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

Execution (TOS) → PC
(SP) + 1 → SP

Status Bits None

Description This instruction replaces the value in the PC with the 16-bit value from the TOS. The SP is incremented by 1. If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 5 cycles
3 cycles (delayed)

Classes Class 32 (see page 3-70)

Example

```
RET
```

	Before Instruction		After Instruction
PC	2112	PC	1000
SP	0300	SP	0301
Data Memory			
0300h	1000	0300h	1000

Syntax **RETE**[**D**]

Operands None

Opcode

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

Execution
(TOS) → PC
(SP) + 1 → SP
0 → INTM

Status Bits Affects INTM

Description This instruction replaces the value in the PC with the 16-bit value from the TOS. Execution continues from this address. The SP is incremented by 1. This instruction automatically clears the interrupt mask bit (INTM) in ST1. (Clearing this bit enables interrupts.) If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

Note:

This instruction is not repeatable.

Words 1 word

Cycles
5 cycles
3 cycles (delayed)

Classes Class 32 (see page 3-70)

Example

```
RETE
```

	Before Instruction		After Instruction
PC	01C3	PC	0110
SP	2001	SP	2002
ST1	xCxx	ST1	x4xx
Data Memory			
2001h	0110	2001h	0110

Syntax **RETF**[**D**]

Operands None

Opcode

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

Execution
(RTN) → PC
(SP) + 1 → SP
0 → INTM

Status Bits Affects INTM

Description This instruction replaces the value in the PC with the 16-bit value in RTN. RTN holds the address to which the interrupt service routine should return. RTN is loaded into the PC during the return instead of reading the PC from the stack. The SP is incremented by 1. This instruction automatically clears the interrupt mask bit (INTM) in ST1. (Clearing this bit enables interrupts.) If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

Note:

You can use this instruction only if no call is performed during the interrupt service routine and no other interrupt routine is taken.

This instruction is not repeatable.

Words 1 word

Cycles
3 cycles
1 cycle (delayed)

Classes Class 33 (see page 3-71)

Example `RETF`

	Before Instruction		After Instruction
PC	01C3	PC	0110
SP	2001	SP	2002
ST1	xCxx	ST1	x4xx
Data Memory			
2001h	0110	2001h	0110

Syntax **RND** *src* [, *dst*]

Operands src , dst: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (src) + 8000h → dst

Status Bits Affected by OVM

Description This instruction rounds the content of *src* (either A or B) by adding 2^{15}. The rounded value is stored in *dst* or *src*, if *dst* is not specified.

> **Note:**
>
> This instruction is not repeatable.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
RND A, B
```

	Before Instruction	After Instruction
A	FF FFFF FFFF	FF FFFF FFFF
B	00 0000 0001	00 0000 7FFF
OVM	0	0

Example 2

```
RND A
```

	Before Instruction	After Instruction
A	00 7FFF FFFF	00 7FFF FFFF
OVM	1	1

Syntax **ROL** *src*

Operands src : A (accumulator A)
B (accumulator B)

Opcode

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

Execution
(C) → src(0)
(src(30–0)) → src(31–1)
(src(31)) → C
0 → src(39–32)

Status Bits Affected by C
Affects C

Description This instruction rotates each bit of *src* left 1 bit. The value of the carry bit, C, before the execution of the instruction is shifted into the LSB of *src*. Then, the MSB of *src* is shifted into C. The guard bits of *src* are cleared.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
ROL A
```

	Before Instruction		After Instruction
A	5F B000 1234	A	00 6000 2468
C	0	C	1

Syntax **ROLTC** *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
(TC) → src(0)
(src(30–0)) → src(31–1)
(src(31)) → C
0 → src(39–32)

Status Bits
Affects C
Affected by TC

Description This instruction rotates each bit of *src* left 1 bit. The value of the TC bit before the execution of the instruction is shifted into the LSB of *src*. Then, the MSB of *src* is shifted into C. The guard bits of *src* are cleared.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
ROLTC A
```

	Before Instruction		After Instruction
A	81 C000 5555	A	00 8000 AAAB
C	x	C	1
TC	1	TC	1

Syntax **ROR** *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution
(C) → src(31)
(src(31–1)) → src(30–0)
(src(0)) → C
0 → src(39–32)

Status Bits
Affects C
Affected by C

Description This instruction rotates each bit of *src* right 1 bit. The value of the carry bit, C, before the execution of the instruction is shifted into the MSB of *src*. Then, the LSB of *src* is shifted into C. The guard bits of *src* are cleared.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
ROR A
```

	Before Instruction		After Instruction
A	7F B000 1235	A	00 5800 091A
C	0	C	1

Syntax

1: **RPT** *Smem*
2: **RPT** *#K*
3: **RPT** *#lk*

Operands

Smem: Single data-memory operand
$0 \leq K \leq 255$
$0 \leq lk \leq 65\,535$

Opcode

1:

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

2:

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

3:

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

Execution

1: (Smem) → RC
2: K → RC
3: lk → RC

Status Bits

None

Description

The repeat counter (RC) is loaded with the number of iterations when this instruction is executed. The number of iterations (n) is given in a 16-bit single data-memory operand *Smem* or an 8- or 16-bit constant, *K* or *lk*, respectively. The instruction following the repeat instruction is repeated n + 1 times. You cannot access RC while it decrements.

Note:

This instruction is not repeatable.

Words

Syntaxes 1 and 2: 1 word
Syntax 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntax 1: 3 cycles
Syntax 2: 1 cycle
Syntax 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 5A (see page 3-9)
Syntax 1: Class 5B (see page 3-9)
Syntax 2: Class 1 (see page 3-3)
Syntax 3: Class 2 (see page 3-4)

Example 1

```
RPT DAT127 ; DAT127 .EQU 0FFF
```

	Before Instruction		After Instruction
RC	0	RC	000C
DP	031	DP	031
Data Memory			
0FFFh	000C	0FFFh	000C

Example 2

```
RPT #2 ; Repeat next instruction 3 times
```

	Before Instruction		After Instruction
RC	0	RC	0002

Example 3

```
RPT #1111h ; Repeat next instruction 4370 times
```

	Before Instruction		After Instruction
RC	0	RC	1111

Syntax **RPTB**[**D**] *pmad*

Operands $0 \leq \text{pmad} \leq 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	Z	0	0	1	1	1	0	0	1	0
16-bit constant															

Execution

$1 \rightarrow$ BRAF
If (delayed) then
    (PC) + 4 $\rightarrow$ RSA
Else
    (PC) + 2 $\rightarrow$ RSA
pmad $\rightarrow$ REA

Status Bits Affects BRAF

Description This instruction repeats a block of instructions the number of times specified by the memory-mapped block-repeat counter (BRC). BRC must be loaded before the execution of this instruction. When this instruction is executed, the block-repeat start address register (RSA) is loaded with PC + 2 (or PC + 4 if you use the delayed instruction) and the block-repeat end address register (REA) is loaded with the program-memory address (*pmad*).

This instruction is interruptible. Single-instruction repeat loops can be included as part of block repeat blocks. To nest block repeat instructions you must ensure that:

- ❑ BRC, RSA, and REA are appropriately saved and restored.
- ❑ The block-repeat active flag (BRAF) is properly set.

In a delayed block repeat (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following this instruction is fetched and executed.

Note:

Block repeat can be deactivated by clearing the BRAF bit.

Far branch and far call instructions cannot be included in a repeat block of instructions.

This instruction is not repeatable.

Words 2 words

Cycles 4 cycles
2 cycles (delayed)

Classes Class 29A (see page 3-66)

Example 1

```
ST #99, BRC
RPTB end_block – 1

      ; end_block = Bottom of Block
```

	Before Instruction		After Instruction
PC	1000	PC	1002
BRC	1234	BRC	0063
RSA	5678	RSA	1002
REA	9ABC	REA	end_block – 1

Example 2

```
ST #99, BRC ;execute the block 100 times
RPTBD end_block – 1
MVDM POINTER, AR1

      ; initialize pointer
      ; end_block ; Bottom of Block
```

	Before Instruction		After Instruction
PC	1000	PC	1004
BRC	1234	BRC	0063
RSA	5678	RSA	1004
REA	9ABC	REA	end_block – 1

Syntax **RPTZ** *dst, #lk*

Operands
dst: A (accumulator A)
B (accumulator B)
$0 \leq lk \leq 65\,535$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	1	1	1	0	0	0	1
16-bit constant															

Execution
$0 \rightarrow dst$
$lk \rightarrow RC$

Status Bits None

Description This instruction clears *dst* and repeats the next instruction n + 1 times, where n is the value in the repeat counter (RC). The RC value is obtained from the 16-bit constant *lk*.

Words 2 words

Cycles 2 cycles

Classes Class 2 (see page 3-4)

Example

```
RPTZ A, 1023 ; Repeat the next instruction 1024 times
STL A, *AR2+
```

	Before Instruction		After Instruction
A	0F FE00 8000	A	00 0000 0000
RC	0000	RC	03FF

Syntax **RSBX** *N*, *SBIT*

Operands $0 \leq \text{SBIT} \leq 15$
$N = 0$ or 1

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	N	0	1	0	1	1	S	B	I	T

Execution $0 \rightarrow \text{STN(SBIT)}$

Status Bits None

Description This instruction clears the specified bit in status register 0 or 1 to a logic 0. *N* designates the status register to modify and *SBIT* specifies the bit to be modified. The name of a field in a status register can be used as an operand instead of the *N* and *SBIT* operands (see Example1).

> **Note:**
>
> This instruction is not repeatable.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
RSBX SXM ; SXM means: n=1 and SBIT=8
```

	Before Instruction		After Instruction
ST1	35CD	ST1	34CD

Example 2

```
RSBX 1,8
```

	Before Instruction		After Instruction
ST1	35CD	ST1	34CD

Syntax **SACCD** *src, Xmem, cond*

Operands

src: A (accumulator A)
B (accumulator B)
Xmem: Dual data-memory operand

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) ≠ 0	0100	BNEQ	(B) ≠ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) ≥ 0	0010	BGEQ	(B) ≥ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) ≤ 0	0111	BLEQ	(B) ≤ 0	1111

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	S	X	X	X	X	C	O	N	D

Execution

If (cond)
Then
(src) << (ASM – 16) → Xmem
Else
(Xmem) → (Xmem)

Status Bits Affected by ASM and SXM

Description If the condition is true, this instruction stores *src* left-shifted by (ASM – 16). The shift value is in the memory location designated by *Xmem*. If the condition is false, the instruction reads *Xmem* and writes the value in *Xmem* back to the same address; thus, *Xmem* remains the same. Regardless of the condition, *Xmem* is always read and updated.

Words 1 word

Cycles 1 cycle

Classes Class 15 (see page 3-32)

Example

```
SACCD A, *AR3+0%, ALT
```

	Before Instruction		After Instruction
A	FF FE00 4321	A	FF FE00 4321
ASM	01	ASM	01
AR0	0002	AR0	0002
AR3	0202	AR3	0204
Data Memory			
0202h	0101	0202h	FC00

Syntax **SAT** *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution Saturate (src) → src

Status Bits Affects OVsrc

Description Regardless of the OVM value, this instruction allows the saturation of the content of *src* on 32 bits.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1 `SAT B`

	Before Instruction		After Instruction
B	71 2345 6789	B	00 7FFF FFFF
OVB	x	OVB	1

Example 2 `SAT A`

	Before Instruction		After Instruction
A	F8 1234 5678	A	FF 8000 0000
OVA	x	OVA	1

Example 3 `SAT B`

	Before Instruction		After Instruction
B	00 0012 3456	B	00 0012 3456
OVB	x	OVB	0

Syntax **SFTA** *src*, *SHIFT* [, *dst*]

Operands src, dst A (accumulator A)
B (accumulator B)
–16 ≤ SHIFT ≤ 15

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	1	1	S	H	I	F	T

Execution

If SHIFT < 0
Then
    (src((–SHIFT) – 1)) → C
    (src(39–0)) << SHIFT → dst
    If SXM = 1
    Then
        (src(39)) → dst(39–(39 + (SHIFT + 1))) [or src(39–(39 + (SHIFT + 1))), if dst is not specified]
    Else
        0 → dst(39–(39 + (SHIFT + 1))) [or src(39–(39 + (SHIFT + 1))), if dst is not specified]
Else
    (src(39 – SHIFT)) → C
    (src) << SHIFT → dst
    0 → dst((SHIFT – 1)–0) [or src((SHIFT – 1)–0), if dst is not specified]

Status Bits Affected by SXM and OVM
Affects C and OVdst (or OVsrc, if dst = src)

Description This instruction arithmetically shifts *src* and stores the result in *dst* or *src*, if *dst* is not specified. The execution of the instruction depends on the SHIFT value:

- If the SHIFT value is less than 0, the following occurs:
  1) *src*((–SHIFT) – 1) is copied into the carry bit, C.
  2) If SXM is 1, the instruction executes an arithmetic right shift and the MSB of the *src* is shifted into *dst*(39–(39 + (SHIFT + 1))).
  3) If SXM is 0, 0 is written into *dst*(39–(39 + (SHIFT + 1))).

- If the SHIFT value is greater than 0, the following occurs:
  1) *src*(39 – SHIFT) is copied into the carry bit, C.
  2) An arithmetic left shift is produced by the instruction.
  3) 0 is written into *dst*((SHIFT – 1)–0).

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
SFTA A, –5, B
```

	Before Instruction		After Instruction
A	FF 8765 0055	A	FF 8765 0055
B	00 4321 1234	B	FF FC3B 2802
C	x	C	1
SXM	1	SXM	1

Example 2

```
SFTA B, +5
```

	Before Instruction		After Instruction
B	80 AA00 1234	B	15 4002 4680
C	0	C	1
OVM	0	OVM	0
SXM	0	SXM	0

Syntax **SFTC** *src*

Operands src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

If (src) = 0
Then
  1 → TC
Else
  If (src(31)) XOR (src(30)) = 0
  Then (two significant sign bits)
    0 → TC
    (src) << 1 → src
  Else (only one sign bit)
    1 → TC

Status Bits Affects TC

Description If *src* has two significant sign bits, this instruction shifts the 32-bit *src* left by 1 bit. If there are two sign bits, the test control (TC) bit is cleared to 0; otherwise, it is set to 1.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
SFTC A
```

	Before Instruction		After Instruction
A	FF FFFF F001	A	FF FFFF E002
TC	x	TC	0

Syntax **SFTL** *src*, *SHIFT* [, *dst*]

Operands src, dst: A (accumulator A)
B (accumulator B)
–16 ≤ SHIFT ≤ 15

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	1	1	1	S	H	I	F	T

Execution

If SHIFT < 0
Then
  src((–SHIFT) – 1) → C
  src(31–0) << SHIFT → dst
  0 → dst(39–(31 + (SHIFT + 1)))
If SHIFT = 0
Then
  0 → C
Else
  src(31 – (SHIFT – 1)) → C
  src((31 – SHIFT)–0) << SHIFT → dst
  0 → dst((SHIFT – 1)–0) [or src((SHIFT – 1)–0), if dst is not specified]
  0 → dst(39–32) [or src(39–32), if dst is not specified]

Status Bits Affects C

Description This instruction logically shifts *src* and stores the result in *dst* or *src*, if *dst* is not specified. The guard bits of *dst* or *src*, if *dst* is not specified, are also cleared. The execution of the instruction depends on the SHIFT value:

- If the SHIFT value is less than 0, the following occurs:

  1) *src*((–SHIFT) – 1) is copied into the carry bit, C.
  2) A logical right shift is produced by the instruction.
  3) 0 is written into *dst*(39–(31 + (SHIFT + 1))).

- If the SHIFT value is greater than 0, the following occurs:

  1) *src*(31 – (SHIFT – 1)) is copied into the carry bit, C.
  2) A logical left shift is produced by the instruction.
  3) 0 is written into *dst*((SHIFT – 1)–0).

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
SFTL A, -5, B
```

	Before Instruction		After Instruction
A	FF 8765 0055	A	FF 8765 0055
B	FF 8000 0000	B	00 043B 2802
C	0	C	1

Example 2

```
SFTL B, +5
```

	Before Instruction		After Instruction
B	80 AA00 1234	B	00 4002 4680
C	0	C	1

Syntax **SQDST** *Xmem*, *Ymem*

Operands Xmem, Ymem: Dual data-memory operands

Opcode

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

Execution (A(32–16)) × (A(32–16)) + (B) → B
((Xmem) – (Ymem)) << 16 → A

Status Bits Affected by OVM, FRCT, and SXM
Affects C, OVA, and OVB

Description Used in repeat single mode, this instruction computes the square of the distance between two vectors. The high part of accumulator A (bits 32–16) is squared, the product is added to accumulator B, and the result is stored in accumulator B. *Ymem* is subtracted from *Xmem*, the difference is shifted 16 bits left, and the result is stored in accumulator A. The value to be squared (A(32–16)) is the value of the accumulator before the subtraction is executed by this instruction.

Words 1 word

Cycles 1 cycle

Classes Class 7 (see page 3-12)

Example

```
SQDST *AR3+, AR4+
```

	Before Instruction		After Instruction
A	FF ABCD 0000	A	FF FFAB 0000
B	00 0000 0000	B	00 1BB1 8229
FRCT	0	FRCT	0
AR3	0100	AR3	0101
AR4	0200	AR4	0201
Data Memory			
0100h	0055	0100h	0055
0200h	00AA	0200h	00AA

Syntax

1: **SQUR** *Smem, dst*
2: **SQUR** **A**, *dst*

Operands

Smem: Single data-memory operand
dst: A (accumulator A)
B (accumulator B)

Opcode

1:

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

2:

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

Execution

1: (Smem) → T
(Smem) × (Smem) → dst
2: (A(32–16)) × (A(32–16)) → dst

Status Bits

Affected by OVM and FRCT
Affects OVsrc

Description

This instruction squares a single data-memory operand *Smem* or the high part of accumulator A (bits 32–16) and stores the result in *dst*. T is unaffected when accumulator A is used; otherwise, *Smem* is stored in T.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntax 2: Class 1 (see page 3-3)

Example 1

```
SQUR 30, B
```

	Before Instruction	After Instruction
B	00 0000 01F4	00 0000 00E1
T	0003	000F
FRCT	0	0
DP	006	006
Data Memory		
031Eh	000F	000F

Example 2

```
SQUR A, B
```

	Before Instruction		After Instruction
A	00 000F 0000	A	00 000F 0000
B	00 0101 0101	B	00 0000 01C2
FRCT	1	FRCT	1

Syntax **SQURA** *Smem, src*

Operands

Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

(Smem) → T
(Smem) × (Smem) + (src) → src

Status Bits

Affected by OVM and FRCT
Affects OVsrc

Description This instruction stores the data-memory value *Smem* in T, then it squares *Smem* and adds the product to *src*. The result is stored in *src*.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example 1

```
SQURA 30, B
```

	Before Instruction		After Instruction
B	00 0320 0000	B	00 0320 00E1
T	0003	T	000F
FRCT	0	FRCT	0
DP	006	DP	006
Data Memory			
031Eh	000F	031Eh	000F

Example 2

```
SQURA *AR3+, A
```

	Before Instruction		After Instruction
A	00 0000 01F4	A	00 0000 02D5
T	0003	T	000F
FRCT	0	FRCT	0
AR3	031E	AR3	031F
Data Memory			
031Eh	000F	031Eh	000F

Syntax **SQURS** *Smem, src*

Operands

Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

(Smem) → T
(src) – (Smem) × (Smem) → src

Status Bits

Affected by OVM and FRCT
Affects OVsrc

Description

This instruction stores the data-memory value *Smem* in T, then it squares *Smem* and subtracts the product from *src*. The result is stored in *src*.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example 1

```
SQURS 9, A
```

	Before Instruction		After Instruction
A	00 014B 5DB0	A	00 0000 0320
T	8765	T	1234
FRCT	0	FRCT	0
DP	006	DP	006
Data Memory			
0309h	1234	0309h	1234

Example 2

```
SQURS *AR3, B
```

	Before Instruction		After Instruction
B	00 014B 5DB0	B	00 0000 0320
T	8765	T	1234
FRCT	0	FRCT	0
AR3	0309	AR3	0309
Data Memory			
0309h	1234	0309h	1234

Syntax **SRCCD** *Xmem*, *cond*

Operands Xmem: Dual data-memory operand

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) ≠ 0	0100	BNEQ	(B) ≠ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) ≥ 0	0010	BGEQ	(B) ≥ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) ≤ 0	0111	BLEQ	(B) ≤ 0	1111

Opcode

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

Execution

If (cond)
Then
  (BRC) → Xmem
Else
  (Xmem) → Xmem

Status Bits None

Description If the condition is true, this instruction stores the content of the block-repeat counter (BRC) in *Xmem*. If the condition is false, the instruction reads *Xmem* and writes the value in *Xmem* back to the same address; thus, *Xmem* remains the same. Regardless of the condition, *Xmem* is always read and updated.

Words 1 word

Cycles 1 cycle

Classes Class 15 (see page 3-32)

Example

```
SRCCD *AR5–, AGT
```

	Before Instruction	After Instruction
A	00 70FF FFFF	00 70FF FFFF
AR5	0202	0201
BRC	4321	4321
Data Memory		
0202h	1234	4321

Syntax **SSBX** *N*, *SBIT*

Operands $0 \leq$ SBIT ≤ 15
N = 0 or 1

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	N	1	1	0	1	1	S	B	I	T

Execution 1 → STN(SBIT)

Status Bits None

Description This instruction sets the specified bit in status register 0 or 1 to a logic 1. *N* designates the status register to modify and *SBIT* specifies the bit to be modified. The name of a field in a status register can be used as an operand instead of the *N* and *SBIT* operands (see Example 1).

> **Note:**
>
> This instruction is not repeatable.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example 1

```
SSBX SXM  ; SXM means: N=1, SBIT=8
```

	Before Instruction		After Instruction
ST1	34CD	ST1	35CD

Example 2

```
SSBX 1,8
```

	Before Instruction		After Instruction
ST1	34CD	ST1	35CD

Syntax

1: **ST T**, *Smem*
2: **ST TRN**, *Smem*
3: **ST** *#lk*, *Smem*

Operands

Smem: Single data-memory operand
–32 768 ≤ lk ≤ 32 767

Opcode

1:

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

2:

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

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	I	A	A	A	A	A	A	A
16-bit constant															

Execution

1: (T) → Smem
2: (TRN) → Smem
3: lk → Smem

Status Bits

None

Description

This instruction stores the content of T, the transition register (TRN), or a 16-bit constant *lk* in data-memory location *Smem*.

Words

Syntaxes 1 and 2: 1 word
Syntax 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1 and 2: 1 cycle
Syntax 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1 and 2: Class 10A (see page 3-22)
Syntaxes 1 and 2: Class 10B (see page 3-23)
Syntax 3: Class 12A (see page 3-26)
Syntax 3: Class 12B (see page 3-27)

Example 1

```
ST FFFFh, 0
```

	Before Instruction		After Instruction
DP	004	DP	004
Data Memory			
0200h	0101	0200h	FFFF

Example 2

```
ST TRN, 5
```

	Before Instruction		After Instruction
DP	004	DP	004
TRN	1234	TRN	1234
Data Memory			
0205h	0030	0205h	1234

Example 3

```
ST T, *AR7–
```

	Before Instruction		After Instruction
T	4210	T	4210
AR7	0321	AR7	0320
Data Memory			
0321h	1200	0321h	4210

Syntax

1: **STH** *src*, *Smem*
2: **STH** *src*, **ASM**, *Smem*
3: **STH** *src*, *SHFT*, *Xmem*
4: **STH** *src* [, *SHIFT*], *Smem*

Operands

src: A (accumulator A)
B (accumulator B)
Smem: Single data-memory operand
Xmem: Dual data-memory operand
$0 \leq \text{SHFT} \leq 15$
$-16 \leq \text{SHIFT} \leq 15$

Opcode

1:

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

2:

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

3:

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

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	0	S	0	1	1	S	H	I	F	T

Execution

1: (src) << (–16) → Smem
2: (src) << (ASM – 16) → Smem
3: (src) << (SHFT – 16) → Xmem
4: (src) << (SHIFT – 16) → Smem

Status Bits

Affected by SXM

Description

This instruction stores the high part of *src* (bits 31–16) in data-memory location *Smem*. The *src* is shifted left (as specified by ASM, SHFT, or SHIFT) and bits 31–16 of the shifted value are stored in data memory (*Smem* or *Xmem*). If SXM = 0, bit 39 of *src* is copied in the MSBs of the data-memory location. If SXM = 1, the sign-extended value with bit 39 of *src* is stored in the MSBs of the data-memory location after being right-shifted by the exceeding guard bit margin. The *src* remains unaffected.

Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- Syntax 3: If *SHFT* = 0, the instruction opcode is assembled as syntax 1.
- Syntax 4: If *SHIFT* = 0, the instruction opcode is assembled as syntax 1.
- Syntax 4: If $0 < SHIFT \leq 15$ and an indirect modifier is equal to one of the Xmem modes, the instruction opcode is assembled as syntax 3.

Words

Syntaxes 1, 2, and 3: 1 word
Syntax 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1, 2, and 3: 1 cycle
Syntax 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1, 2, and 3: Class 10A (see page 3-22)
Syntaxes 1 and 2: Class 10B (see page 3-23)
Syntax 4: Class 11A (see page 3-24)
Syntax 4: Class 11B (see page 3-25)

Example 1

```
STH A, 10
```

	Before Instruction		After Instruction
A	FF 8765 4321	A	FF 8765 4321
DP	004	DP	004
Data Memory			
020Ah	1234	020Ah	8765

Example 2

```
STH B, –8, *AR7–
```

	Before Instruction		After Instruction
B	FF 8421 1234	B	FF 8421 1234
AR7	0321	AR7	0320
Data Memory			
0321h	ABCD	0321h	FF84

Example 3

```
STH A, -4, 10
```

	Before Instruction		After Instruction
A	FF 8421 1234	A	FF 8421 1234
SXM	1	SXM	1
DP	004	DP	004
Data Memory			
020Ah	7FFF	020Ah	F842

Syntax

1: **STL** *src*, *Smem*
2: **STL** *src*, **ASM**, *Smem*
3: **STL** *src*, *SHFT*, *Xmem*
4: **STL** *src* [, *SHIFT*], *Smem*

Operands

src: A (accumulator A)
B (accumulator B)
Smem: Single data-memory operand
Xmem: Dual data-memory operand
$0 \leq$ SHFT ≤ 15
$-16 \leq$ SHIFT ≤ 15

Opcode

1:

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

2:

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

3:

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

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	0	S	1	0	0	S	H	I	F	T

Execution

1: (src) → Smem
2: (src) << ASM → Smem
3: (src) << SHFT → Xmem
4: (src) << SHIFT → Smem

Status Bits

Affected by SXM

Description

This instruction stores the low part of *src* (bits 15–0) in data-memory location *Smem*. The *src* is shifted left (as specified by ASM, SHFT, or SHIFT) and bits 15–0 of the shifted value are stored in data memory (*Smem* or *Xmem*). When the shifted value is positive, zeros are shifted into the LSBs.

Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- Syntax 3: If *SHFT* = 0, the instruction opcode is assembled as syntax 1.
- Syntax 4: If *SHIFT* = 0, the instruction opcode is assembled as syntax 1.
- Syntax 4: If 0 $<$ *SHIFT* $\leq$ 15 and an indirect modifier is equal to one of the Xmem modes, the instruction opcode is assembled as syntax 3.

Words

Syntaxes 1, 2, and 3: 1 word
Syntax 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1, 2, and 3: 1 cycle
Syntax 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1, 2, and 3: Class 10A (see page 3-22)
Syntaxes 1, 2, and 3: Class 10B (see page 3-23)
Syntax 4: Class 11A (see page 3-24)
Syntax 4: Class 11B (see page 3-25)

Example 1

```
STL A, 11
```

	Before Instruction	After Instruction
A	FF 8765 4321	FF 8765 4321
DP	004	004
Data Memory		
020Bh	1234	4321

Example 2

```
STL B, –8, *AR7–
```

	Before Instruction	After Instruction
B	FF 8421 1234	FF 8421 1234
SXM	0	0
AR7	0321	0320
Data Memory		
0321h	0099	2112

Example 3

```
STL A, 7, 11
```

	Before Instruction		After Instruction
A	FF 8421 1234	A	FF 8421 1234
DP	004	DP	004
Data Memory			
020Bh	0101	020Bh	1A00

Syntax **STLM** *src*, *MMR*

Operands

src: A (accumulator A)
B (accumulator B)
MMR: Memory-mapped register

Opcode

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

Execution (src(15–0)) → MMR

Status Bits None

Description This instruction stores the low part of *src* (bits 15–0) into the addressed memory-mapped register *MMR*. The nine MSBs of the effective address are cleared to 0 regardless of the current value of DP or of the upper nine bits of ARx. This instruction allows *src* to be stored in any memory location on data page 0 without modifying the DP field in status register ST0.

Words 1 word

Cycles 1 cycle

Classes Class 10A (see page 3-22)

Example 1

```
STLM A, BRC
```

	Before Instruction		After Instruction
A	FF 8765 4321	A	FF 8765 4321
BRC(1Ah)	1234	BRC	4321

Example 2

```
STLM B, *AR1–
```

	Before Instruction		After Instruction
B	FF 8421 1234	B	FF 8421 1234
AR1	3F17	AR1	0016
AR7(17h)	0099	AR7	1234

Syntax **STM** *#lk*, *MMR*

Operands MMR: Memory-mapped register
$-32\,768 \leq lk \leq 32\,767$

Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	I	A	A	A	A	A	A	A
16-bit constant															

Execution lk → MMR

Status Bits None

Description This instruction stores a 16-bit constant *lk* into a memory-mapped register *MMR* or a memory location on data page 0 without modifying the DP field in status register ST0. The nine MSBs of the effective address are cleared to 0 regardless of the current value of DP or of the upper nine bits of ARx.

Words 2 words

Cycles 2 cycles

Classes Class 12A (see page 3-26)

Example 1

```
STM 0FFFFh, IMR
```

	Before Instruction	After Instruction
IMR	FF01	FFFF

Example 2

```
STM 8765h, *AR7+
```

	Before Instruction	After Instruction
AR0	0000	8765
AR7	8010	0011

Syntax

ST *src*, *Ymem*
|| ADD *Xmem*, *dst*

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Xmem, Ymem:	Dual data-memory operands
dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A

Opcode

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

Execution

(src) << (ASM − 16) → Ymem
(dst_) + (Xmem) << 16 → dst

Status Bits

Affected by OVM, SXM, and ASM
Affects C and OVdst

Description

This instruction stores *src* shifted by (ASM – 16) in data-memory location *Ymem*. In parallel, this instruction adds the content of *dst_* to the data-memory operand *Xmem* shifted left 16 bits, and stores the result in *dst*. If *src* is equal to *dst*, the value stored in *Ymem* is the value of *src* before the execution.

Words

1 word

Cycles

1 cycle

Classes

Class 14 (see page 3-30)

Example

```
ST A, *AR3
||ADD *AR5+0%, B
```

	Before Instruction	After Instruction
A	FF 8421 1000	FF 8021 1000
B	00 0000 1111	FF 0422 1000
OVM	0	0
SXM	1	1
ASM	1	1
AR0	0002	0002
AR3	0200	0200
AR5	0300	0302
Data Memory		
0200h	0101	0842
0300h	8001	8001

Syntax

1: **ST** *src*, *Ymem*
|| **LD** *Xmem*, *dst*

2: **ST** *src*, *Ymem*
|| **LD** *Xmem*, **T**

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Xmem, Ymem:	Dual data-memory operands

Opcode

1:

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

2:

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

Execution

1. (src) << (ASM – 16) → Ymem
   (Xmem) << 16 → dst
2. (src) << (ASM – 16) → Ymem
   (Xmem) → T

Status Bits

Affected by OVM and ASM
Affects C

Description

This instruction stores *src* shifted by (ASM – 16) in data-memory location *Ymem*. In parallel, this instruction loads the 16-bit dual data-memory operand *Xmem* to *dst* or T. If *src* is equal to *dst*, the value stored in *Ymem* is the value of *src* before the execution.

Words

1 word

Cycles

1 cycle

Classes

Class 14 (see page 3-30)

Example 1

```
ST B, *AR2–
||LD *AR4+, A
```

	Before Instruction		After Instruction
A	00 0000 001C	A	FF 8001 0000
B	FF 8421 1234	B	FF 8421 1234
SXM	1	SXM	1
ASM	1C	ASM	1C
AR2	01FF	AR2	01FE
AR4	0200	AR4	0201
Data Memory			
01FFh	xxxx	01FFh	F842
0200h	8001	0200h	8001

Example 2

```
ST A, *AR3
||LD *AR4, T
```

	Before Instruction		After Instruction
A	FF 8421 1234	A	FF 8421 1234
T	3456	T	80FF
ASM	1	ASM	1
AR3	0200	AR3	0200
AR4	0100	AR4	0100
Data Memory			
0200h	0001	0200h	0842
0100h	80FF	0100h	80FF

Example 3

```
ST A, *AR2+
||LD *AR2–, A
```

In Example 3, the LD reads the source operand at the memory location pointed to by AR2 before the ST writes to the same location. The ST reads the source operand of accumulator A before LD loads accumulator A.

Syntax

ST *src*, *Ymem*
|| MAC[**R**] *Xmem*, *dst*

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Xmem, Ymem:	Dual data-memory operands

Opcode

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

Execution

(src << (ASM – 16)) → Ymem
If (Rounding)
Then
Round ((Xmem) × (T) + (dst)) → dst
Else
(Xmem) × (T) + (dst) → dst

Status Bits

Affected by OVM, SXM, ASM, and FRCT
Affects C and OVdst

Description

This instruction stores *src* shifted by (ASM – 16) in data-memory location *Ymem*. In parallel, this instruction multiplies the content of T by the data-memory operand *Xmem*, adds the value in *dst* (with or without rounding), and stores the result in *dst*. If *src* is equal to *dst*, the value stored in *Ymem* is the value of *src* before the execution of this instruction.

If you use the R suffix, this instruction rounds the result of the multiply accumulate operation by adding 2^{15} to the result and clearing the LSBs (bits 15–0) to 0.

Words

1 word

Cycles

1 cycle

Classes

Class 14 (see page 3-30)

Example 1

```
ST A, *AR4–
||MAC *AR5, B
```

	Before Instruction	After Instruction
A	00 0011 1111	00 0011 1111
B	00 0000 1111	00 010C 9511
T	0400	0400
ASM	5	5
FRCT	0	0
AR4	0100	00FF
AR5	0200	0200
Data Memory		
100h	1234	0222
200h	4321	4321

Example 2

```
ST A, *AR4+
||MACR *AR5+, B
```

	Before Instruction	After Instruction
A	00 0011 1111	00 0011 1111
B	00 0000 1111	00 010D 0000
T	0400	0400
ASM	1C	1C
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
100h	1234	0001
200h	4321	4321

Syntax

ST *src*, *Ymem*
|| MAS[**R**] *Xmem*, *dst*

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Xmem, Ymem:	Dual data-memory operands

Opcode

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

Execution

(src << (ASM − 16)) → Ymem
If (Rounding)
  Then
  Round ((dst) – (Xmem) × (T))→ dst
Else
  (dst) – (Xmem) × (T) → dst

Status Bits

Affected by OVM, SXM, ASM, and FRCT
Affects C and OVdst

Description

This instruction stores *src* shifted by (ASM – 16) in data-memory location *Ymem*. In parallel, this instruction multiplies the content of T by the data-memory operand *Xmem*, subtracts the value from *dst* (with or without rounding), and stores the result in *dst*. If *src* is equal to *dst*, the value stored in *Ymem* is the value of *src* before the execution of this instruction.

If you use the R suffix, this instruction optionally rounds the result of the multiply subtract operation by adding 2^{15} to the result and clearing the LSBs (bits 15–0) to 0.

Words

1 word

Cycles

1 cycle

Classes

Class 14 (see page 3-30)

Example 1

```
ST A, *AR4+
||MAS *AR5, B
```

	Before Instruction		After Instruction
A	00 0011 1111	A	00 0011 1111
B	00 0000 1111	B	FF FEF3 8D11
T	0400	T	0400
ASM	5	ASM	5
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0200
Data Memory			
0100h	1234	0100h	0222
0200h	4321	0200h	4321

Example 2

```
ST A, *AR4+
||MASR *AR5+, B
```

	Before Instruction		After Instruction
A	00 0011 1111	A	00 0011 1111
B	00 0000 1111	B	FF FEF4 0000
T	0400	T	0400
ASM	1	ASM	1
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	1234	0100h	0022
0200h	4321	0200h	4321

Syntax

ST *src*, *Ymem*
|| MPY *Xmem*, *dst*

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Xmem, Ymem:	Dual data-memory operands

Opcode

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

Execution

(src << (ASM − 16)) → Ymem
(T) × (Xmem) → dst

Status Bits

Affected by OVM, SXM, ASM, and FRCT
Affects C and OVdst

Description

This instruction stores *src* shifted by (ASM – 16) in data-memory location *Ymem*. In parallel, this instruction multiplies the content of T by the 16-bit dual data-memory operand *Xmem*, and stores the result in *dst*. If *src* is equal to *dst*, then the value stored in *Ymem* is the value of *src* before the execution.

Words

1 word

Cycles

1 cycle

Classes

Class 14 (see page 3-30)

Example

```
ST A, *AR3+
||MPY *AR5+, B
```

	Before Instruction	After Instruction
A	FF 8421 1234	FF 8421 1234
B	xx xxxx xxxx	00 2000 0000
T	4000	4000
ASM	00	00
FRCT	1	1
AR3	0200	0201
AR5	0300	0301
Data Memory		
0200h	1111	8421
0300h	4000	4000

Syntax

ST *src*, *Ymem*
|| SUB *Xmem*, *dst*

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Xmem, Ymem:	Dual data-memory operands
dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A.

Opcode

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

Execution

(src << (ASM – 16)) → Ymem
(Xmem) << 16 – (dst_) → dst

Status Bits

Affected by OVM, SXM, and ASM
Affects C and OVdst

Description

This instruction stores *src* shifted by (ASM – 16) in data-memory location *Ymem*. In parallel, this instruction subtracts the content of *dst_* from the 16-bit dual data-memory operand *Xmem* shifted left 16 bits, and stores the result in *dst*. If *src* is equal to *dst*, then the value stored in *Ymem* is the value of *src* before the execution.

Words

1 word

Cycles

1 cycle

Classes

Class 14 (see page 3-30)

Example

```
ST A, *AR3–
||SUB *AR5+0%, B
```

	Before Instruction	After Instruction
A	FF 8421 0000	FF 8421 0000
B	00 1000 0001	FF FBE0 0000
ASM	01	01
SXM	1	1
AR0	0002	0002
AR3	01FF	01FE
AR5	0300	0302
Data Memory		
01FFh	1111	0842
0300h	8001	8001

Syntax **STRCD** *Xmem*, *cond*

Operands Xmem: Dual data-memory operand

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) ≠ 0	0100	BNEQ	(B) ≠ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) ≥ 0	0010	BGEQ	(B) ≥ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) ≤ 0	0111	BLEQ	(B) ≤ 0	1111

Opcode

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

Execution If (cond)
(T) → Xmem
Else
(Xmem) → Xmem

Status Bits None

Description If the condition is true, this instruction stores the content of T into the data-memory location *Xmem.* If the condition is false, the instruction reads *Xmem* and writes the value in *Xmem* back to the same address; thus, *Xmem* remains the same. Regardless of the condition, *Xmem* is always read and updated.

Words 1 word

Cycles 1 cycle

Classes Class 15 (see page 3-32)

Example

```
STRCD *AR5–, AGT
```

	Before Instruction	After Instruction
A	00 70FF FFFF	00 70FF FFFF
T	4321	4321
AR5	0202	0201
Data Memory		
0202h	1234	4321

Syntax

1: **SUB** *Smem, src*
2: **SUB** *Smem*, **TS**, *src*
3: **SUB** *Smem*, **16**, *src* [, *dst*]
4: **SUB** *Smem* [, *SHIFT*], *src* [, *dst*]
5: **SUB** *Xmem, SHFT, src*
6: **SUB** *Xmem, Ymem, dst*
7: **SUB** *#lk* [, *SHFT*], *src* [, *dst*]
8: **SUB** *#lk*, **16**, *src* [, *dst*]
9: **SUB** *src* [, *SHIFT*], [, *dst*]
10: **SUB** *src*, **ASM** [, *dst*]

Operands

src, dst: A (accumulator A)
B (accumulator B)
Smem: Single data-memory operand
Xmem, Ymem: Dual data-memory operands
$-32\,768 \leq lk \leq 32\,767$
$0 \leq SHFT \leq 15$
$-16 \leq SHIFT \leq 15$

Opcode

1:

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

2:

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

3:

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

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	S	D	0	0	1	S	H	I	F	T

5:

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

6:

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

7:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	0	0	1	S	H	F	T
16-bit constant															

8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	0	0	1
16-bit constant															

9:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	0	1	S	H	I	F	T

10:

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

Execution

1: (src) – (Smem) → src
2: (src) – (Smem) << TS → src
3: (src) – (Smem) << 16 → dst
4: (src) – (Smem) << SHIFT → dst
5: (src) – (Xmem) << SHFT → src
6: (Xmem) << 16 – (Ymem) << 16 → dst
7: (src) – lk << SHFT → dst
8: (src) – lk << 16 → dst
9: (dst) – (src) << SHIFT → dst
10: (dst) – (src) << ASM → dst

Status Bits

Affected by SXM and OVM
Affects C and OVdst (or OVsrc, if dst = src)

For instruction syntax 3, if the result of the subtraction generates a borrow, the carry bit, C, is cleared to 0; otherwise, C is not affected.

Description

This instruction subtracts a 16-bit value from the content of the selected accumulator or from the 16-bit operand *Xmem* in dual data-memory addressing mode. The 16-bit value to be subtracted is one of the following:

- The content of a single data-memory operand (*Smem*)
- The content of a dual data-memory operand (*Ymem*)
- A 16-bit immediate operand (#*lk*)
- The shifted value in *src*

If a *dst* is specified, this instruction stores the result in *dst*. If no *dst* is specified, this instruction stores the result in *src*. Most of the second operands can be shifted. For a left shift:

- Low-order bits are cleared
- High-order bits are:
  - Sign extended if SXM = 1
  - Cleared if SXM = 0

For a right shift, the high-order bits are:

- Sign extended if SXM = 1
- Cleared if SXM = 0

Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- Syntax 4: If *dst* = *src* and *SHIFT* = 0, then the instruction opcode is assembled as syntax 1.
- Syntax 4: If *dst* = *src*, *SHIFT* ≤ 15, and *Smem* indirect addressing mode is included in *Xmem*, then the instruction opcode is assembled as syntax 1.

Words

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 word
Syntaxes 4, 7, and 8: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 cycle
Syntaxes 4, 7, and 8: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntaxes 1, 2, 3, and 5: Class 3A (see page 3-5)
Syntaxes 1, 2, and 3: Class 3B (see page 3-6)
Syntax 4: Class 4A (see page 3-7)
Syntax 4: Class 4B (see page 3-8)
Syntax 6: Class 7 (see page 3-12)
Syntaxes 7 and 8: Class 2 (see page 3-4)
Syntaxes 9 and 10: Class 1 (see page 3-3)

Example 1

```
SUB *AR1+, 14, A
```

	Before Instruction	After Instruction
A	00 0000 1200	FF FAC0 1200
C	x	0
SXM	1	1
AR1	0100	0101
Data Memory		
0100h	1500	1500

Example 2

```
SUB A, −8, B
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	00 0000 17EE
C	x	C	1
SXM	1	SXM	1

Example 3

```
SUB #12345, 8, A, B
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	FF FFCF D900
C	x	C	0
SXM	1	SXM	1

Syntax **SUBB** *Smem, src*

Operands

src: A (accumulator A)
B (accumulator B)

Smem: Single data-memory operand

Opcode

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

Execution (src) – (Smem) – (logical inversion of C) → src

Status Bits

Affected by OVM and C
Affects C and OVsrc

Description This instruction subtracts the content of the 16-bit single data-memory operand *Smem* and the logical inverse of the carry bit, C, from *src* without sign extension.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example 1

```
SUBB 5, A
```

	Before Instruction	After Instruction
A	00 0000 0006	FF FFFF FFFF
C	0	0
DP	008	008
Data Memory		
0405h	0006	0006

Example 2

```
SUBB *AR1+, B
```

	Before Instruction	After Instruction
B	FF 8000 0006	FF 8000 0000
C	1	1
OVM	1	1
AR1	0405	0406
Data Memory		
0405h	0006	0006

Syntax **SUBC** *Smem, src*

Operands

Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution

(src) – ((Smem) << 15) → ALU output
If ALU output ≥ 0
Then
((ALU output) << 1) + 1 → src
Else (src) << 1 → src

Status Bits

Affected by SXM
Affects C and OVsrc

Description

This instruction subtracts the 16-bit single data-memory operand *Smem*, left-shifted 15 bits, from the content of *src*. If the result is greater than 0, it is shifted 1 bit left, 1 is added to the result, and the result is stored in *src*. Otherwise, this instruction shifts the content of *src* 1 bit left and stores the result in *src*.

The divisor and the dividend are both assumed to be positive in this instruction. The SXM bit affects this operation in these ways:

- If SXM = 1, the divisor must have a 0 value in the MSB.
- If SXM = 0, any 16-bit divisor value produces the expected results.

The dividend, which is in *src*, must initially be positive (bit 31 must be 0) and must remain positive following the accumulator shift, which occurs in the first portion of the instruction.

This instruction affects OVA or OVB (depending on *src*) but is not affected by OVM; therefore, *src* does not saturate on positive or negative overflows when executing this instruction.

Words

1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example 1

```
SUBC 2, A
```

	Before Instruction		After Instruction
A	00 0000 0004	A	00 0000 0008
C	x	C	0
DP	006	DP	006
Data Memory			
0302h	0001	0302h	0001

Example 2

```
RPT #15
SUBC *AR1, B
```

	Before Instruction		After Instruction
B	00 0000 0041	B	00 0002 0009
C	x	C	1
AR1	1000	AR1	1000
Data Memory			
1000h	0007	1000h	0007

Syntax **SUBS** *Smem, src*

Operands

Smem: Single data-memory operand
src: A (accumulator A)
B (accumulator B)

Opcode

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

Execution (src) – unsigned (Smem) → src

Status Bits Affected by OVM
Affects C and OVsrc

Description This instruction subtracts the content of the 16-bit single data-memory operand *Smem* from the content of *src*. *Smem* is considered a 16-bit unsigned number regardless of the value of SXM. The result is stored in *src*.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 3A (see page 3-5)
Class 3B (see page 3-6)

Example

```
SUBS *AR2–, B
```

	Before Instruction	After Instruction
B	00 0000 0002	FF FFFF 0FFC
C	x	0
AR2	0100	00FF
Data Memory		
0100h	F006	F006

Syntax **TRAP** *K*

Operands $0 \leq K \leq 31$

Opcode

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

Execution

(SP) − 1 → SP
(PC) + 1 → TOS
Interrupt vector specified by K → PC

Status Bits None

Description This instruction transfers program control to the interrupt vector specified by *K*. This instruction allows you to use your software to execute any interrupt service routine. For a list of interrupts and their corresponding *K* value, see your device datasheet.

This instruction pushes PC + 1 onto the data-memory location addressed by SP. This enables a return instruction to retrieve the pointer to the instruction after the trap from the data-memory location addressed by SP. This instruction is not maskable and is not affected by INTM nor does it affect INTM.

Note:

This instruction is not repeatable.

Words 1 word

Cycles 3 cycles

Classes Class 35 (see page 3-72)

Example

```
TRAP 10h
```

	Before Instruction		After Instruction
PC	1233	PC	FFC0
SP	03FF	SP	03FE
Data Memory			
03FEh	9653	03FEh	1234

Syntax **WRITA** *Smem*

Operands Smem: Single data-memory operand

Opcode

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

Execution

A → PAR
If (RC) ≠ 0
Then
  (Smem) → (Pmem addressed by PAR)
  (PAR) + 1 → PAR
  (RC) – 1 → RC
Else
  (Smem) → (Pmem addressed by PAR)

Status Bits None

Description This instruction transfers a word from a data-memory location specified by *Smem* to a program-memory location. The program-memory location is defined by accumulator A, depending on the specific device, as follows:

C541–C546	Devices with Extended Program Memory
A(15–0)	A(22–0)

This instruction can be used with the repeat instruction to move consecutive words (using indirect addressing) in data memory to a continuous program-memory space addressed by PAR by automatically incrementing PAR. The initial value is set with the 16 LSBs of accumulator A. The source and destination blocks in memory do not have to be entirely on-chip or off-chip. When used with repeat, this instruction becomes a single-cycle instruction once the repeat pipeline is started.

The content of accumulator A is not affected by this instruction.

Words 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 5 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 26A (see page 3-60)
Class 26B (see page 3-62)

Example `WRITA 5`

	Before Instruction		After Instruction
A	00 0000 0257	A	00 0000 0257
DP	032	DP	032
Program Memory			
0257h	0306	0257h	4339
Data Memory			
1005h	4339	1005h	4339

Syntax **XC** *n*, *cond* [, *cond* [, *cond*]]

Operands n = 1 or 2

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{BIO}$ low	0000 0011	NBIO	$\overline{BIO}$ high	0000 0010
C	C = 1	0000 1100	NC	C = 0	0000 1000
TC	TC = 1	0011 0000	NTC	TC = 0	0010 0000
AEQ	(A) = 0	0100 0101	BEQ	(B) = 0	0100 1101
ANEQ	(A) ≠ 0	0100 0100	BNEQ	(B) ≠ 0	0100 1100
AGT	(A) > 0	0100 0110	BGT	(B) > 0	0100 1110
AGEQ	(A) ≥ 0	0100 0010	BGEQ	(B) ≥ 0	0100 1010
ALT	(A) < 0	0100 0011	BLT	(B) < 0	0100 1011
ALEQ	(A) ≤ 0	0100 0111	BLEQ	(B) ≤ 0	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Opcode

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

Syntax n	Opcode N
1	0
2	1

Execution If (cond)
Then
Next n instructions are executed
Else
Execute NOP for next n instructions

Status Bits None

Description

The execution of this instruction depends on the value of n and the selected conditions:

- If n = 1 and the condition(s) is met, the 1-word instruction following this instruction is executed.
- If n = 2 and the condition(s) is met, the one 2-word instruction or the two 1-word instructions following this instruction are executed.
- If the condition(s) is not met, one or two nops are executed depending on the value of n.

This instruction tests multiple conditions before executing and can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

Group 1: You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time. The accumulator must be the same for both conditions; you cannot test conditions for both accumulators with the same instruction. For example, you can test AGT and AOV at the same time, but you cannot test AGT and BOV at the same time.

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

Conditions for This Instruction

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

This instruction and the two instruction words following this instruction are uninterruptible.

> **Note:**
>
> The conditions tested are sampled two full cycles before this instruction is executed. Therefore, if the two 1-word instructions or one 2-word instruction modifies the conditions, there is no effect on the execution of this instruction, but if the conditions are modified during the two slots, the interrupt operation using this instruction can cause undesirable results.
>
> This instruction is not repeatable.

Words 1 word

Cycles 1 cycle

Classes Class 1 (see page 3-3)

Example

```
XC 1, ALEQ
MAR *AR1+
ADD A, DAT100
```

	Before Instruction		After Instruction
A	FF FFFF FFFF	A	FF FFFF FFFF
AR1	0032	AR1	0033

If the content of accumulator A is less than or equal to 0, AR1 is modified before the execution of the addition instruction.

Syntax

1: **XOR** *Smem, src*
2: **XOR** *#lk* [, *SHFT*], *src* [, *dst*]
3: **XOR** *#lk,* **16**, *src* [, *dst*]
4: **XOR** *src* [, *SHIFT*] [, *dst*]

Operands

src, dst: A (accumulator A)
B (accumulator B)
Smem: Single data-memory operand
$0 \leq$ SHFT ≤ 15
$-16 \leq$ SHIFT ≤ 15
$0 \leq$ lk $\leq 65\,535$

Opcode

1:

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

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	0	1	S	H	F	T
16-bit constant															

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	1	0	1
16-bit constant															

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	1	1	0	S	H	I	F	T

Execution

1: (Smem) XOR (src) → src
2: lk << SHFT XOR (src) → dst
3: lk << 16 XOR (src) → dst
4: (src) << SHIFT XOR (dst) → dst

Status Bits

None

Description

This instruction executes an exclusive OR of the 16-bit single data-memory operand *Smem* (shifted as indicated in the instruction) with the content of the selected accumulator and stores the result in *dst* or *src*, as specified. For a left shift, the low-order bits are cleared and the high-order bits are not sign extended. For a right shift, the sign is not extended.

Words

Syntaxes 1 and 4: 1 word
Syntaxes 2 and 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles

Syntaxes 1 and 4: 1 cycle
Syntaxes 2 and 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes

Syntax 1: Class 3A (see page 3-5)
Syntax 1: Class 3B (see page 3-6)
Syntaxes 2 and 3: Class 2 (see page 3-4)
Syntax 4: Class 1 (see page 3-3)

Example 1

```
XOR *AR3+, A
```

	Before Instruction		After Instruction
A	00 00FF 1200	A	00 00FF 0700
AR3	0100	AR3	0101
Data Memory			
0100h	1500	0100h	1500

Example 2

```
XOR A, +3, B
```

	Before Instruction		After Instruction
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	00 0000 8800

Syntax **XORM** *#lk, Smem*

Operands Smem: Single data-memory operand
$0 \leq lk \leq 65\,535$

Opcode

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

Execution lk XOR (Smem) → Smem

Status Bits None

Description This instruction executes an exclusive OR of the content of a data-memory location *Smem* with a 16-bit constant *lk*. The result is written to *Smem*.

Note:

This instruction is not repeatable.

Words 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

Cycles 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

Classes Class 18A (see page 3-39)
Class 18B (see page 3-39)

Example

```
XORM 0404h, *AR4–
```

	Before Instruction		After Instruction
AR4	0100	AR4	00FF
Data Memory			
0100h	4444	0100h	4040

Appendix A

Condition Codes

This appendix lists the conditions for conditional instructions (Table A–1) and the combination of conditions that can be tested (Table A–2). Conditional instructions can test conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

Group1: You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time. The accumulator must be the same for both conditions; you cannot test conditions for both accumulators with the same instruction. For example, you can test AGT and AOV at the same time, but you cannot test AGT and BOV at the same time.

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

Table A–1. Conditions for Conditional Instructions

Operand	Condition	Description
AEQ	A = 0	Accumulator A equal to 0
BEQ	B = 0	Accumulator B equal to 0
ANEQ	$A \neq 0$	Accumulator A not equal to 0
BNEQ	$B \neq 0$	Accumulator B not equal to 0
ALT	A < 0	Accumulator A less than 0
BLT	B < 0	Accumulator B less than 0
ALEQ	$A \leq 0$	Accumulator A less than or equal to 0
BLEQ	$B \leq 0$	Accumulator B less than or equal to 0
AGT	A > 0	Accumulator A greater than 0
BGT	B > 0	Accumulator B greater than 0
AGEQ	$A \geq 0$	Accumulator A greater than or equal to 0
BGEQ	$B \geq 0$	Accumulator B greater than or equal to 0
AOV[†]	AOV = 1	Accumulator A overflow detected
BOV[†]	BOV = 1	Accumulator B overflow detected
ANOV[†]	AOV = 0	No accumulator A overflow detected
BNOV[†]	BOV = 0	No accumulator B overflow detected
C[†]	C = 1	ALU carry set to 1
NC[†]	C = 0	ALU carry clear to 0
TC[†]	TC = 1	Test/Control flag set to 1
NTC[†]	TC = 0	Test/Control flag cleared to 0
BIO[†]	$\overline{BIO}$ low	$\overline{BIO}$ signal is low
NBIO[†]	$\overline{BIO}$ high	$\overline{BIO}$ signal is high
UNC[†]	none	Unconditional operation

[†] Cannot be used with conditional store instructions

Table A–2. Groupings of Conditions

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

Appendix B

CPU Status and Control Registers

This appendix shows the bit fields of the TMS320C54x™ CPU status and control registers. The C54x™ DSP has three status and control registers:

- ❑ Status register 0 (ST0)
- ❑ Status register 1 (ST1)
- ❑ Processor mode status register (PMST)

ST0 and ST1 contain the status of various conditions and modes; PMST contains memory-setup status and control information. Because these registers are memory-mapped, they can be stored into and loaded from data memory; the status of the processor can be saved and restored for subroutines and interrupt service routines (ISRs).

Table B–1 defines terms used in identifying the register fields.

Table B–1. Register Field Terms and Definitions

Term	Definition
ARP	Auxiliary register pointer
ASM	Accumulator shift mode
AVIS	Address visibility mode
BRAF	Block repeat active flag
C	Carry
CLKOFF	CLOCKOUT off
CMPT	Compatibility mode
CPL	Compiler mode
C16	Dual 16-bit/double-precision arithmetic mode
DP	Data page pointer
DROM	Data ROM
FRCT	Fractional mode

Table B–1. Register Field Terms and Definitions (Continued)

Term	**Definition**
HM	Hold mode
INTM	Interrupt mode
IPTR	Interrupt vector pointer
MP/$\overline{MC}$	Microprocessor/microcomputer
OVA	Overflow flag A
OVB	Overflow flag B
OVLY	RAM overlay
OVM	Overflow mode
SMUL	Saturation on multiplication
SST	Saturation on store
SXM	Sign-extension mode
TC	Test/control flag
XF	External flag status

Figure B–1. Processor Mode Status Register (PMST)

15–7	6	5	4	3	2	1	0
IPTR	MP/$\overline{MC}$	OVLY	AVIS	DROM	CLKOFF†	SMUL†	SST†

† These bits are only supported on C54x devices with revision A or later, or on C54x devices numbered C548 or greater. You may also refer to the device-specific data sheet to determine if these bits are supported.

Figure B–2. Status Register 0 (ST0)

15–13	12	11	10	9	8–0
ARP	TC	C	OVA	OVB	DP

Figure B–3. Status Register 1 (ST1)

15	14	13	12	11	10	9	8	7	6	5	4–0
BRAF	CPL	XF	HM	INTM	0	OVM	SXM	C16	FRCT	CMPT	ASM

Appendix C

Glossary

A

A: See *accumulator A*.

accumulator: A register that stores the results of an operation and provides an input for subsequent arithmetic logic unit (ALU) operations.

accumulator A: One of two 40-bit registers that store the result of an operation and provide an input for subsequent arithmetic logic unit (ALU) operations.

accumulator B: One of two 40-bit registers that store the result of an operation and provide an input for subsequent arithmetic logic unit (ALU) operations.

accumulator shift mode bits (ASM): A 5-bit field in ST1 that specifies a shift value (from –16 to 15) that is used to shift an accumulator value when executing certain instructions, such as instructions with parallel loads and stores.

address: The location of a word in memory.

address visibility mode bit (AVIS): A bit in PMST that determines whether or not the internal program address appears on the device's external address bus pins.

addressing mode: The method by which an instruction calculates the location of an object in memory.

AG: *Accumulator guard bits*. An 8-bit register that contains bits 39–32 (the guard bits) of an accumulator. Both accumulator A and accumulator B have guards bits.

AH: *Accumulator A high word*. Bits 31–16 of accumulator A.

AL: *Accumulator A low word*. Bits15–0 of accumulator A.

ALU: *Arithmetic logic unit*. The part of the CPU that performs arithmetic and logic operations.

AR0–AR7: See *auxiliary registers*.

ARAU: See *auxiliary register arithmetic unit*.

ARP: See *auxiliary register pointer*.

ASM: See *accumulator shift mode bits*.

auxiliary register arithmetic unit (ARAU): An unsigned, 16-bit arithmetic logic unit (ALU) used to calculate indirect addresses using auxiliary registers.

auxiliary register file: The area in data memory containing the eight 16-bit auxiliary registers. See also *auxiliary registers*.

auxiliary register pointer (ARP): A 3-bit field in ST0 used as a pointer to the currently-selected auxiliary register, when the device is operating in ’C5x/’C2xx compatibility mode.

auxiliary registers (AR0–AR7): Eight 16-bit registers that are used as pointers to an address within data space. These registers are operated on by the auxiliary register arithmetic units (ARAUs) and are selected by the auxiliary register pointer (ARP). See also *auxiliary register arithmetic unit*.

AVIS: See *address visibility mode bit*.

B

B: See *accumulator B*.

barrel shifter: A unit that rotates bits in a word.

BG: *Accumulator B guard bits*. An 8-bit register that contains bits 39–32 (the guard bits) of accumulator B.

BH: *Accumulator B high word*. Bits 31–16 of accumulator B.

BL: *Accumulator B low word*. Bits 15–0 of accumulator B.

block-repeat active flag (BRAF): A 1-bit flag in ST1 that indicates whether or not a block repeat is currently active.

block-repeat counter (BRC): A 16-bit register that specifies the number of times a block of code is to be repeated when a block repeat is performed.

block-repeat end address register (REA): A 16-bit memory-mapped register containing the end address of a code segment being repeated.

block-repeat start address register (RSA): A 16-bit memory-mapped register containing the start address of a code segment being repeated.

boot: The process of loading a program into program memory.

boot loader: A built-in segment of code that transfers code from an external source to program memory at power-up.

BRC: See *block-repeat counter*.

butterfly: A kernel function for computing an N-point fast Fourier transform (FFT), where N is a power of 2. The combinational pattern of inputs resembles butterfly wings.

C

C16: A bit in ST1 that determines whether the ALU operates in dual 16-bit mode or in double-precision mode.

CAB: *C address bus*. A bus that carries addresses needed for accessing data memory.

carry bit (C): A bit used by the ALU in extended arithmetic operations and accumulator shifts and rotates. The carry bit can be tested by conditional instructions.

CB: *C bus*. A bus that carries operands that are read from data memory.

CMPT: See *compatibility mode bit*.

code: A set of instructions written to perform a task.

cold boot: The process of loading a program into program memory at power-up.

compatibility mode bit (CMPT): A bit in ST1 that determines whether or not the auxiliary register pointer (ARP) is used to select an auxiliary register in single indirect addressing mode.

compiler mode bit (CPL): A bit in ST1 that determines whether the CPU uses the data page pointer or the stack pointer to generate data memory addresses in direct addressing mode.

CPL: See *compiler mode bit*.

D

DAB: *D address bus*. A bus that carries addresses needed for accessing data memory.

DAB address register (DAR): A register that holds the address to be put on the DAB to address data memory for reads via the DB.

DAGEN: See *data address generation logic*.

DAR: See *DAB address register*.

DARAM: *Dual-access RAM*. Memory that can be accessed twice in the same clock cycle.

data address bus: A group of connections used to route data memory addresses. The C54x DSP has three 16-bit buses that carry data memory addresses: CAB, DAB, and EAB.

data address generation logic (DAGEN): Logic circuitry that generates the addresses for data memory reads and writes. See also *program address generation logic.*

data bus: A group of connections used to route data. The C54x DSP has three 16-bit data buses: CB, DB, and EB.

data memory: A memory region used for storing and manipulating data. Addresses 00h–1Fh of data memory contain CPU registers. Addresses 20h–5Fh of data memory contain peripheral registers.

data page pointer (DP): A 9-bit field in ST0 that specifies which of 512 128-word pages is currently selected for direct address generation. DP provides the nine MSBs of the data-memory address; the data memory address provides the lower seven bits. See also *direct memory address.*

data ROM bit (DROM): A bit in processor mode status register (PMST) that determines whether part of the on-chip ROM is mapped into program space.

DB: *D bus*. A bus that carries operands that are read from data memory.

direct memory address (dma, DMA): The seven LSBs of a direct-addressed instruction that are concatenated with the data page pointer (DP) to generate the entire data memory address. See also *data page pointer.*

dma: *See direct memory address*.

DP: See *data page pointer.*

DROM: See *data ROM bit*.

E

EAB address register (EAR): A register that holds the address to be put on the EAB to address data memory for reads via the EB.

EAR: See *EAB address register*.

EB: *E bus*. A bus that carries data to be written to memory.

exponent (EXP) encoder: A hardware device that computes the exponent value of the accumulator.

F

fast return register (RTN): A 16-bit register used to hold the return address for the fast return from interrupt (RETF[D]) instruction.

fractional mode bit (FRCT): A bit in status register ST1 that determines whether or not the multiplier output is left-shifted by one bit.

FRCT: See *fractional mode bit*.

H

HM: See *hold mode bit.*

hold mode bit (HM): A bit in status register ST1 that determines whether the CPU enters the hold state in normal mode or concurrent mode.

I

IFR: See *interrupt flag register*.

IMR: See *interrupt mask register*.

instruction register (IR): A 16-bit register used to hold a fetched instruction.

interrupt: A condition caused by internal hardware, an event external to the CPU, or by a previously executed instruction that forces the current program to be suspended and causes the processor to execute an interrupt service routine corresponding to the interrupt.

interrupt flag register (IFR): A 16-bit memory-mapped register used to identify and clear active interrupts.

interrupt mask register (IMR): A 16-bit memory-mapped register used to enable or disable external and internal interrupts. A 1 written to any IMR bit position enables the corresponding interrupt (when INTM = 0).

interrupt mode bit (INTM): A bit in status register ST1 that globally masks or enables all interrupts.

interrupt service routine (ISR): A module of code that is executed in response to a hardware or software interrupt.

INTM: See *interrupt mode bit*.

IPTR: *Interrupt vector pointer*. A 9-bit field in the processor mode status register (PMST) that points to the 128-word page where interrupt vectors reside.

IR: See *instruction register.*

ISR: See *interrupt service routine*.

L

latency: The delay between when a condition occurs and when the device reacts to the condition. Also, in a pipeline, the delay between the execution of two instructions that is necessary to ensure that the values used by the second instruction are correct.

LSB: *Least significant bit*. The lowest order bit in a word.

M

memory-mapped register (MMR): The C54x DSP processor registers mapped into page 0 of the data memory space.

microcomputer mode: A mode in which the on-chip ROM is enabled and addressable.

microprocessor mode: A mode in which the on-chip ROM is disabled.

micro stack: A stack that provides temporary storage for the address of the next instruction to be fetched when the program address generation logic is used to generate sequential addresses in data space.

MP/$\overline{\text{MC}}$ bit: A bit in the processor mode status register (PMST) that indicates whether the processor is operating in microprocessor or microcomputer mode. See also *microcomputer mode; microprocessor mode.*

MSB: *Most significant bit*. The highest order bit in a word.

O

OVA: *Overflow flag A*. A bit in status register ST0 that indicates the overflow condition of accumulator A.

OVB: *Overflow flag B*. A bit status register ST0 that indicates the overflow condition of accumulator B.

overflow: A condition in which the result of an arithmetic operation exceeds the capacity of the register used to hold that result.

overflow flag (OVA, OVB): A flag that indicates whether or not an arithmetic operation has exceeded the capacity of the corresponding accumulator. See also *OVA* and *OVB*.

overflow mode bit (OVM): A bit in status register ST1 that specifies how the ALU handles an overflow after an operation.

OVLY: See *RAM overlay bit*.

OVM: See *overflow mode bit.*

P

PAB: *Program address bus.* A 16-bit bus that provides the address for program memory reads and writes.

PAGEN: See *program address generation logic*.

PAR: See *program address register*.

PB: *Program bus*. A bus that carries the instruction code and immediate operands from program memory.

PC: See *program counter.*

pipeline: A method of executing instructions in an assembly-line fashion.

pmad: *Program-memory address.* A16-bit immediate program-memory address.

PMST: See *processor mode status register*.

pop: Action of removing a word from a stack.

processor mode status register (PMST): A 16-bit status register that controls the memory configuration of the device. See also *ST0; ST1.*

program address generation logic (PAGEN): Logic circuitry that generates the address for program memory reads and writes, and the address for data memory in instructions that require two data operands. This circuitry can generate one address per machine cycle. See also *data address generation logic.*

program address register (PAR): A register that holds the address to be put on the PAB to address memory for reads via the PB.

program controller: Logic circuitry that decodes instructions, manages the pipeline, stores status of operations, and decodes conditional operations.

program counter (PC): A 16-bit register that indicates the location of the next instruction to be executed.

program counter extension register (XPC): A register that contains the upper 7 bits of the current program memory address.

program data bus (PB): A bus that carries the instruction code and immediate operands from program memory.

program memory: A memory region used for storing and executing programs.

push: Action of placing a word onto a stack.

R

RAM overlay bit (OVLY): A bit in the processor mode status register PMST that determines whether or not on-chip dual-access RAM is mapped into the program/data space.

RC: See *repeat counter*.

REA: See *block-repeat end address*.

register: A group of bits used for temporarily holding data or for controlling or specifying the status of a device.

repeat counter (RC): A 16-bit register used to specify the number of times a single instruction is executed.

reset: A means of bringing the CPU to a known state by setting the registers and control bits to predetermined values and signaling execution to start at a specified address.

RSA: See *block-repeat start address*.

RTN: See *fast return register*.

S

SARAM: *Single-access RAM.* Memory that only can be read from or written during one clock cycle.

shifter: A hardware unit that shifts bits in a word to the left or to the right.

sign-control logic: Circuitry used to extend data bits (signed/unsigned) to match the input data format of the multiplier, ALU, and shifter.

sign extension: An operation that fills the high order bits of a number with the sign bit.

sign-extension mode bit (SXM): A bit in status register ST1 that enables sign extension in CPU operations.

SINT: See *software interrupt.*

software interrupt: An interrupt caused by the execution of an INTR or TRAP instruction.

SP: See *stack pointer.*

ST0: *Status register 0.* A 16-bit register that contains C54x CPU status and control bits. See also *PMST; ST1.*

ST1: *Status register 1.* A16-bit register that contains C54x CPU status and control bits. See also *PMST; ST0.*

stack: A block of memory used for storing return addresses for subroutines and interrupt service routines and for storing data.

stack pointer (SP): A register that always points to the last element pushed onto the stack.

SXM: See *sign-extension mode bit.*

T

TC: See *test/control flag bit.*

temporary register (T): A 16-bit register that holds one of the operands for multiply and store instructions, the dynamic shift count for the add and subtract instructions, or the dynamic bit position for the bit test instructions.

test/control flag bit (TC): A bit in status register ST0 that is affected by test operations.

transition register (TRN): A 16-bit register that holds the transition decision for the path to new metrics to perform the Viterbi algorithm.

W

warm boot: The process by which the processor transfers control to the entry address of a previously-loaded program.

X

XF: *XF status flag*. A bit in status register ST1 that indicates the status of the XF pin.

XPC: See *program counter extension register.*

Z

ZA: *Zero detect bit A*. A signal that indicates when accumulator A contains a 0.

ZB: *Zero detect bit B*. A signal that indicates when accumulator B contains a 0.

zero detect: See *ZA* and *ZB*.

zero fill: A method of filling the low- or high-order bits with zeros when loading a 16-bit number into a 32-bit field.

Index

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S