Texas Instruments

TAS3xxx DSP Instruction Set

PurePath Digital™ Audio Processors

Reference Guide

January 2009 Performance Media

TEXAS INSTRUMENTS

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Read This First

About This Manual

This manual describes the operation and instruction set of the Digital Signal Processor Core inside the TAS3xxx Digital Audio Processor Devices.

How to Use This Manual

This document contains the following chapters:

Chapter 1 – Overview of the TAS3xxx Audio Digital Signal Processor (DSP) Architecture

Chapter 2 – TAS3xxx DSP Data and Coefficient Formats

Chapter 3 – General Flow for Processing Digital Audio

Chapter 4 – TAS3xxx DSP Instruction Set

This manual is intended to be used by the developer to as a reference guide to programming DSP assembly language in any TAS3xxx Device from Texas Instruments. It is recommended to read through this manual as written for a firm understanding of the DSP core and its associated assembly language instruction set.

Additional Documentation

- TAS3xxx Data Manuals
- General TAS3xxx Application Notes
- PurePath Studio™ Graphical Development Environment User's Guide
- TAS3xxx Firmware Programmers Reference Guide

Trademarks

PurePath Studio™ are trademarks of Texas Instruments

All other Trademarks are property of their respective owners

Figures

Chapter 1

TAS3xxx Audio DSP Architecture Overview

1.1 TAS3xxx DSP Architecture Components:

The TAS3xxx Digital Signal Processor Core is a fixed-point computational engine consisting of an arithmetic unit, a single cycle hardware multiplier, a 76 bit accumulator, data, coefficient memory, and delay memory blocks. The architecture is optimized for programming digital audio processing blocks such as IIR filters, FIR filters, volume control, tone controls, mixers, DRC, loudness, etc. The TAS3xxx DSP Core contains the following components:

- Data RAM: Used as temporary storage for digital audio data
- Coefficient RAM: Contains filter or gain coefficients that can be multiplied by audio data
- Register B: Storage for data to be transformed by one of the following operations: barrel shifting, negation, absolute value, or pass through operation
- Register L: Storage for data to be transformed by one of the following operations: Log_2, Anti-Log_2, negation, absolute value, or pass through operation
- Register MD: Storage for 48 bit data to be multiplied
- Register MC: Storage for 28 bit coefficient data to be multiplied
- Register BR: Storage for result of operations performed on the contents of Register B
- Register LR: Storage for result of operations performed on the contents of Register L
- Register MR: Storage for result of multiplication of data contained in register MD and MC
- ACC: 76 bit Accumulator
- Register DI: Digital Input data from serial audio port (SAP) input or analog to digital converter (ADC) inputs.Data Ram memory mapped so that the lower three Least Significant bits decode the input channels
- Register DO1-DO8: An 8 register block for outputting data to the TAS3xxx Output SAP, or to digital to analog converters (DAC).
- Register LFS: The two least significant bits of the Digital input data that provide the programmer with a way to dither the audio (used in conjunction with Register B)
- Register DLYI: Input register used as an interface to the Delay RAM which provides a mechanism for delaying the audio data
- Register DLYO: Output register from delay RAM

Figure 1-1 shows the TAS3xxx DSP audio processing architecture.

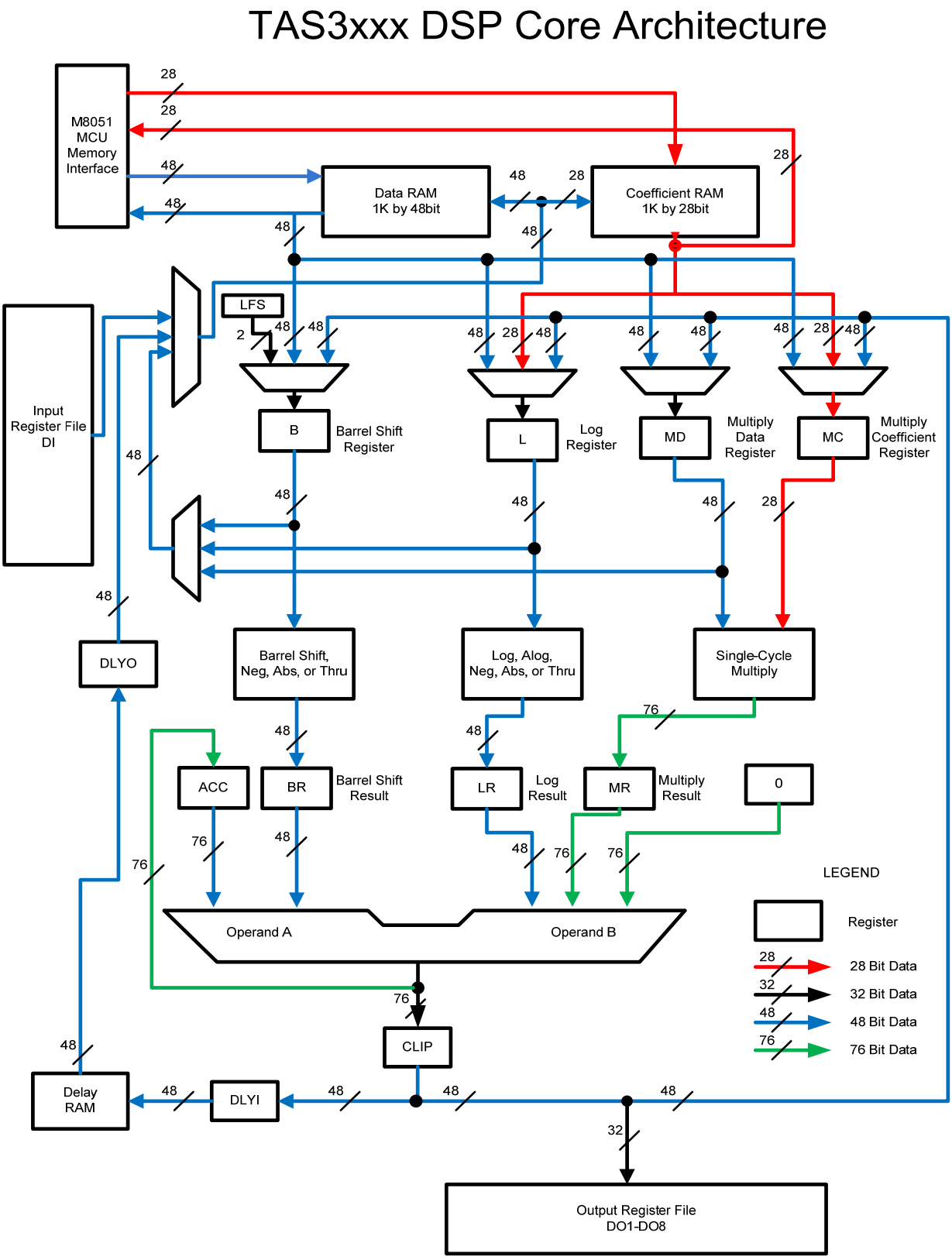


Figure 1-1 TAS3xxx DSP Core Architecture

Chapter 2

DSP Data Number Formats

2.1 Arithmetic Logical Unit Operand Audio Encoding

Figure 2-1 shows the input data word structure of the arithmetic logic unit (ALU). Eight bits of overhead or guard bits are concatenated to the upper end of the 24 bit digital audio data word, and sixteen bits of computational precision or noise bits are concatenated to the lower end of the 24 bit digital audio data word. The incoming digital data words are positioned with the most significant bit abutting the 8-bit overhead/guard boundary regardless of if the digital data is represented by 16, 20, or 24 bits. The sign bit, bit 39, indicates that all incoming audio samples are treated as signed data samples.

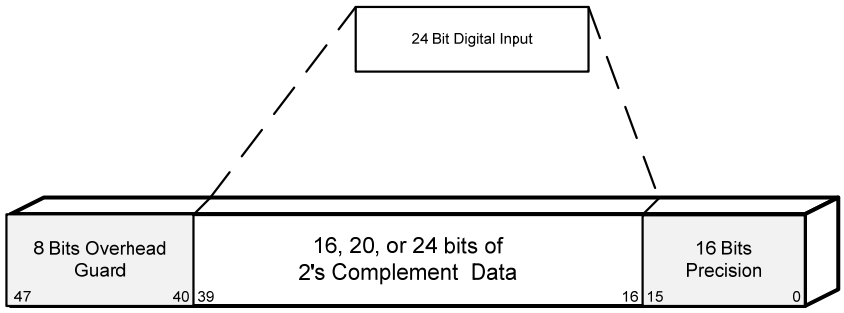


Figure 2-1: Input Data Word Structure of ALU Unit

The arithmetic engine is a 48-bit (25.23 format) processor consisting of a general-purpose 76-bit arithmetic logic unit and function specific arithmetic blocks. Multiply operations (excluding the function-specific arithmetic blocks) always involve 48-bit words and 28-bit coefficients. (The 28 bit coefficients are typically user defined and programmed via the I^2C bus) If a group of products is to be added together, the 76-bit product of each multiplication is applied to a 76-bit adder, where a DSP-like multiply-accumulate operation takes place. Additionally, biquad filter computations use the Multiply-Accumulate operation to maintain precision in the intermediate computational stages.

The memory banks include a dual-port Data RAM for storing intermediate results, a Coefficient RAM for storing filter coefficients, a delay RAM for delaying the processed audio, and a fixed program ROM.

To maximize the linear range of the 76-bit ALU, saturation logic is used. In multiply-accumulate computations, intermediate overflows are permitted, and it is assumed that subsequent terms in the computation flow correct the overflow condition. This logic is detailed below in Figure 2-2.

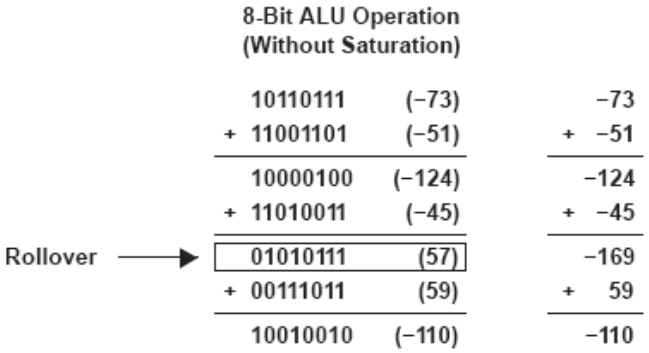


Figure 2-2: Example of simplified TAS3xxx ALU Addition

2.2 Digital Audio Output Number Formats in TAS3xxx DSP Core

The Digital Audio Processor is a 48-bit signed fixed point arithmetic processing machine. The computed data is in 2's complement format with the most significant bit being the sign bit and the lower 47 bits being the data bits. Mixer gain operations are implemented by multiplying a 48-bit signed data value by a 28-bit signed gain coefficient. The 76-bit signed product is then truncated to a signed 48-bit number. Add operations are implemented by a concatenating 28 bit signed offset coefficient to a 48-bit signed data value. In most cases, if the addition results in overflowing the 48-bit signed number format, saturation logic is used. This means that if the summation results in a positive number that is greater that 0x7FFF FFFF FFFF, the number is set to 0x7FFF FFFF FFFF. Likewise, if the summation results in a negative number that is less that 0x8000 0000 0000, the number is set to 0x8000 0000,

2.2.1 28 Bit 5.23 Number Format

All mixer gain coefficients are 28-bit coefficients using a 5.23 number format. Numbers formatted as 5.23 numbers have 5 bits to the left of the implied binary point and 23 bits to the right of the implied binary point. This is shown in the figure 2-3

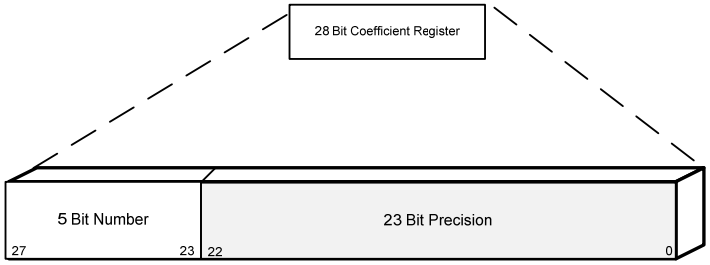


Figure 2-3: 28 Bit 5.23 Number Format

The decimal value of the 5.23 format number can be found by following the weighting as illustrated in figure 2-4. If the most significant bit is logic 0, the number is a positive number, and the weighting shown yields the correct number. If the most significant bit is a logic 1, then the number is a negative number. In this case, every bit must be inverted, a 1 added to the result, and then the weighting shown in Figure 2-4 applied to obtain the magnitude of the negative number.

$$\text{Coefficient Decimal Value} = 2^{4} + 2^{3} + 2^{2} + 2^{1} + 2^{0} + 2^{-1} + 2^{-2} + \ldots + 2^{-22} + 2^{-23}$$

Figure 2-4: 5.23 to Decimal Number Conversion Formula

Gain coefficients, entered via the I^2C bus, must be entered as 32-bit binary numbers. The format of the 32-bit number is shown below.

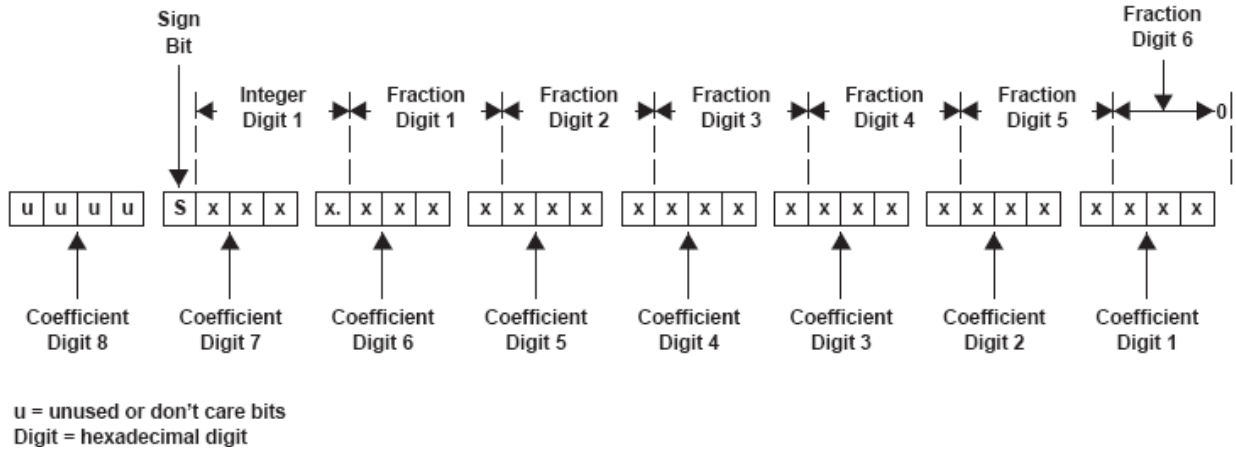


Figure 2-5: I^2C Format for Coefficients

As the above figure shows, the hexadecimal value of the integer part of the gain coefficient cannot be concatenated with the hex value of the fractional part of the gain coefficient to form the 32-bit I^2C coefficient. The reason is that the 28-bit coefficient contains five bits of integer, and thus the integer part of the coefficient occupies all the one hex digit and the most significant bit of the second hex digit. In the same way, the fractional part occupies the lower three bits of the second hex digit, and then occupies the other five hex digits (with the eight digit being the zero-valued most significant hex digit.

2.2.2 48-Bit 25.23 Number Format

All level adjustment and threshold coefficients are 48-bit coefficients using the 25.23 number format. Numbers formatted as 25.23 numbers have 25 bits to the left of the implied decimal point and 23 bits to the right of the implied decimal point. This format is shown in figure 2-6.

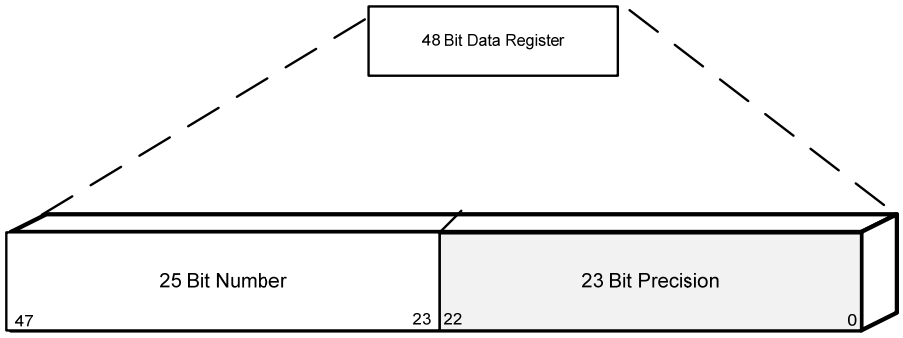


Figure 2-6: 48 Bit 25.23 Number Format

The decimal value of the 25.23 format number can be found by following the weighting shown in the figure 2-7 below. If the most significant bit is logic 0, the number is a positive number, and the weighting shown yields the correct number. If the most significant bit is a logic 1, then the number is a negative number. In this case, every bit must be inverted, a 1 added to the result, and then the weighting shown in Figure 2-7 applied to obtain the magnitude of the negative number.

$$\text{Data Decimal Value} = 2^{24} + 2^{23} + 2^{22} + 2^{21} + 2^{20} + 2^{19} + \ldots 2^{1} + 2^{0} + 2^{-1} \ldots + 2^{-22} + 2^{-23}$$

Figure 2-7:25.23 to Decimal Number Conversion Formula

Two 32 bit words must be sent over the I^2C bus to download a level or threshold coefficient into the TAS3xxx Device. The alignment of the 48 bit, 25.23 formatted coefficient in the 8 byte I^2C word is shown in figure 2-8.

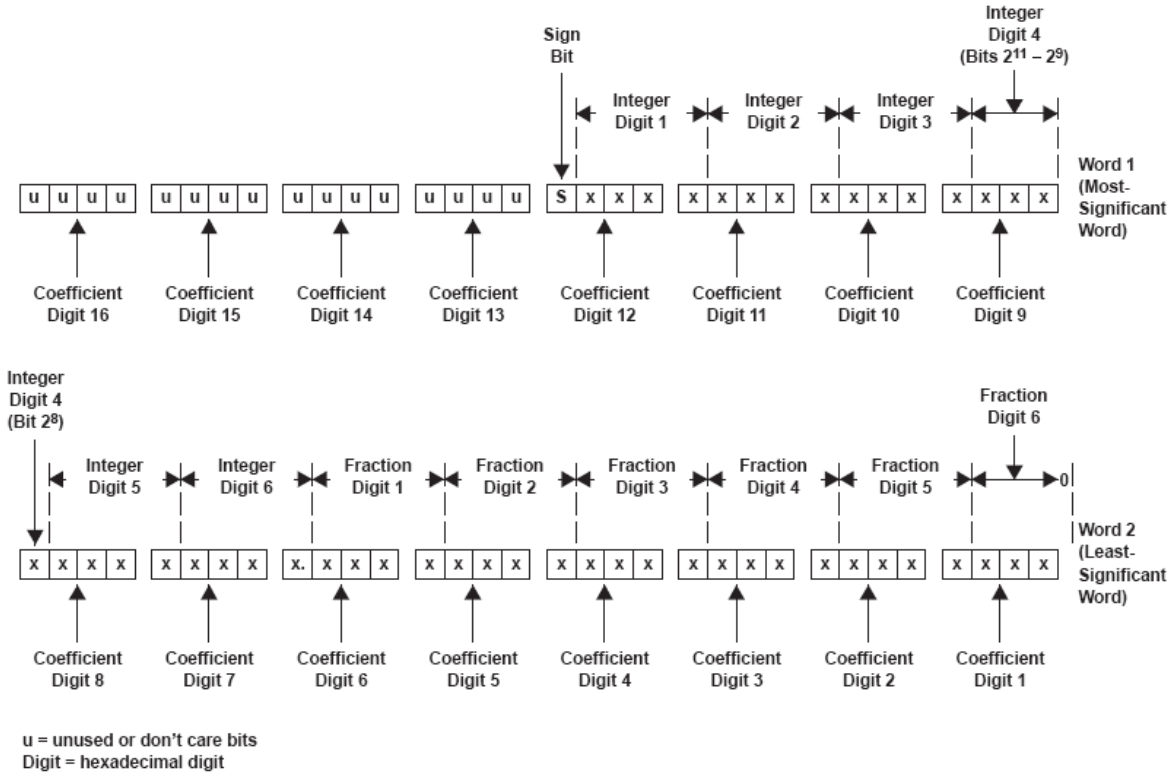


Figure 2-8: I^2C Format for Data

2.2.3 TAS3xxx Digital Audio Scaling

The TAS3xxx digital audio processing is designed so that noise produced by filter operations is maintained below the smallest signal amplitude of interest, as shown in the figure below. This low noise level is achieved by increasing the prevision of the signal representation substantially above the number of bits that are absolutely necessary to represent the input signal.

Additional precision, in the form of overflow bits, is used to permit the value of intermediate calculations to exceed the input precision without clipping. The TAS3xxx Digital Audio Processor achieved both of these important performance capabilities by using a high performance digital audio processing architecture with a 48 bit data path, 28 bit filter coefficients, and a 76 bit Accumulator.

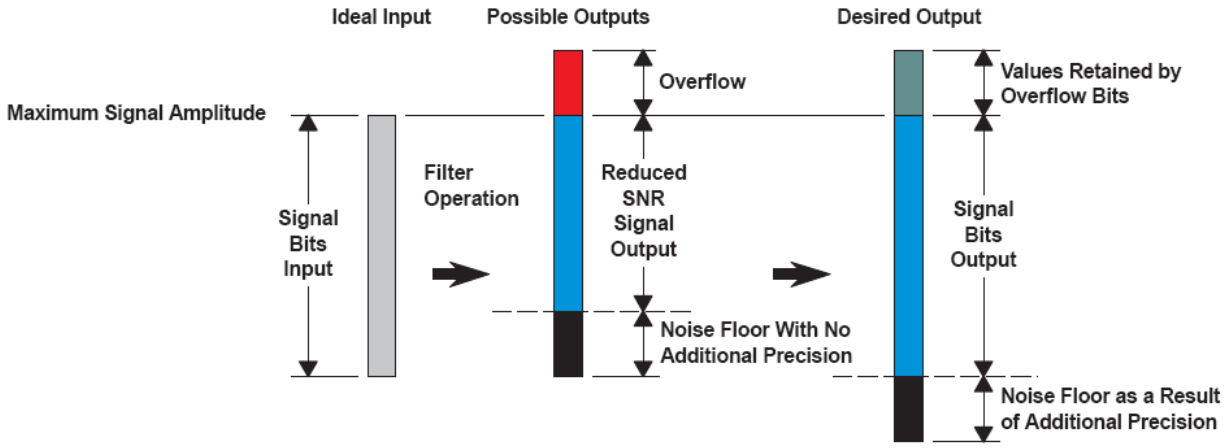


Figure 2-9: TAS3xxx Digital Audio Scaling

2.3 DSP Program Counter

The DSP program counter (PC) is automatically set to 0 on startup and on completion of an LRCLK (F_s) frame. The PC is advanced by the DSP clock, whose frequency is dependent on the sample rate (F_s) and the serial data mode.

For a standard I^2S input at F_s = 48 kHz, and Master Clock (MCLK) of 12.288 MHZ, the DSP clock runs at approximately 135 MHz. This means that there are ~2816 DSP cycles per sample available for processing.

Chapter 3

3.1 Audio Processing Flow

The TAS3xxx audio processing is normally done on a sample-by-sample bases at rate equal to the sampling frequency F_s. The general flow of the TAS3xxx processing scheme is shown in Figure 3-1

1. Digital audio data from either Serial Audio Port (SAP) or Analog-to-Digital Converter (ADC) is input
2. Digital audio data is processed by DSP
3. Processed digital audio is output to SAP, Digital to Analog Converter, or S/PDIF
4. Real-time control from user is provided by I^2C using embedded M8051 Microcontroller Unit (MCU)

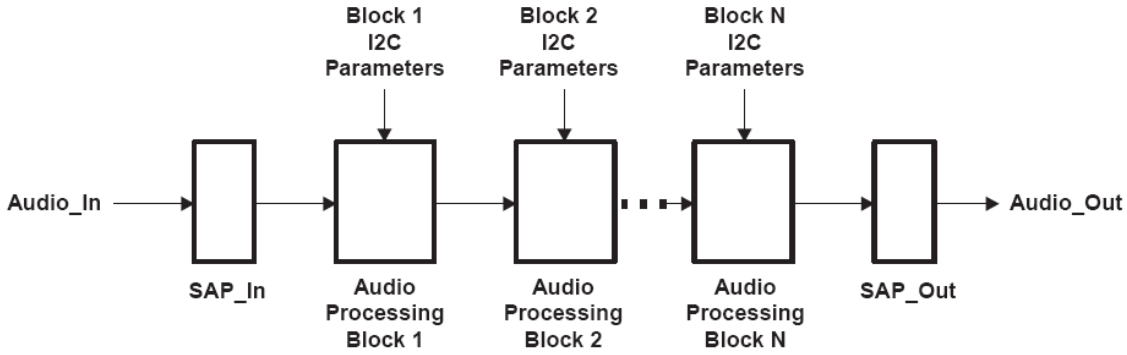


Figure 3-1.General TAS3xxx Audio Process Flow

3.1.1 Digital Audio Data Input

The TAS3xxx Family of Devices offer both digital audio data inputs (SDIN1, SDIN2, etc…) and Analog-to-Digital converted audio data. In most applications, these audio inputs are stereo inputs from an external source. The digital data is input to the DSP through Register DI. These inputs can be immediately processed, or stored in Data RAM for later processing in the cycle.

Note: Refer to the respective TAS3xxx Data Manuals for exact input and output options available.

3.1.2 Audio Data Processing

Immediately after the data leaves Register DI, it is normally processed by a series of blocks programmed in TAS3xxx DSP assembly language. For the non-experienced DSP assembly language programmer, a high level graphical drag and drop tool is available from Texas Instruments. For more information about this tool, refer to the PurePath Studio™ Graphical Development Environment User's Guide. These processing blocks are normally common audio blocks such as IIR filters, volume control, tone control, Dynamic Range Compression (DRC), loudness, delay, mixers, and other commonly used blocks. Because the TAS3xxx DSP is fully programmable, the only limitation to the number of processing blocks is the available hardware resources (DSP Program RAM, DSP Coefficient RAM, and DSP Data RAM) and the developer's imagination.

A simple example of an audio processing block is shown in Figure 3-2. This example is a 4 input mixer that provides real-time control of the mixer gains by way of the I^2C coefficients GAIN1, GAIN2, GAIN3, and GAIN4

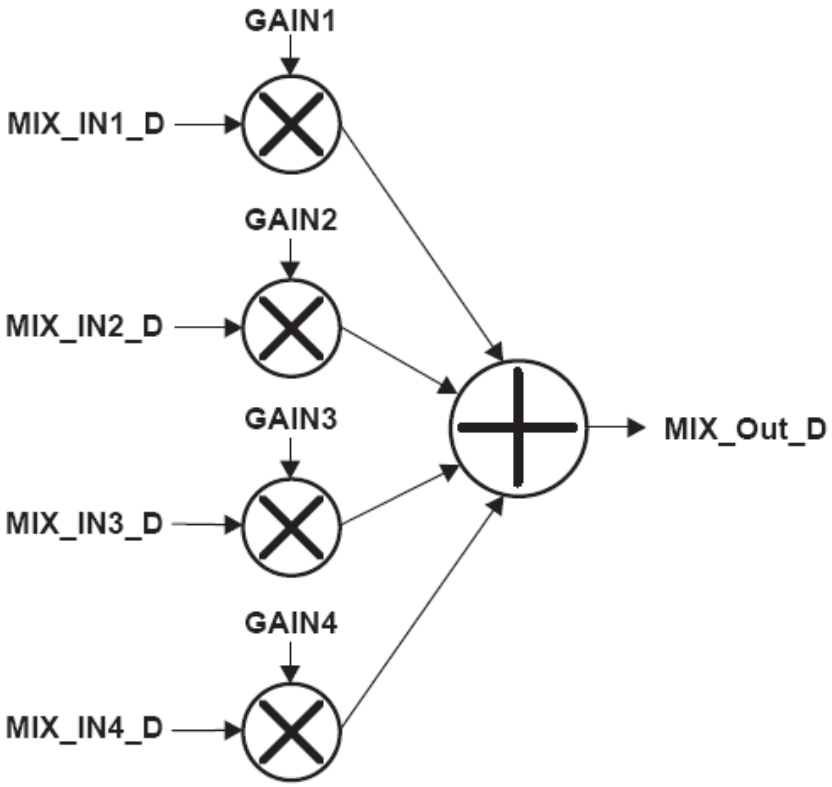


Figure 3-2: Example Audio Processing Block Diagram

An example of the TAS3xxx assembly code used to implement this four input mixer is shown in Table 3-1. It is assumed that the Inputs to the processing block (MIX_IN1_D, MIX_IN2_D, MIX_IN3_D, and MIX_IN4_D) have had the previous block outputs stored to them prior to this example. Likewise, the data output from this block (MIX_OUT_D) is available for further processing after the completion of this block.

ALU1	ALU2	Data RAM Load to Register	COEF RAM Load to Register	RAM Store
NOP |	CLRACC |	LD(MIX_IN1_D,MD) |	LD(GAIN1,MC) |	NOP
NOP |	NOP |	LD(MIX_IN2_D,MD) |	LD(GAIN2,MC) |	NOP
NOP |	ADD(ACC,MR,NONE,ACC) |	LD(MIX_IN3_D,MD) |	LD(GAIN3,MC) |	NOP
NOP |	ADD(ACC,MR,NONE,ACC) |	LD(MIX_IN4_D,MD) |	LD(GAIN4,MC) |	NOP
NOP |	ADD(ACC,MR,NONE,ACC) |	NOP |	NOP |	NOP
NOP |	ADD(ACC,MR,NONE,B) |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	ST(B,DATA,MIX_OUT_D)

Table 3-1: Example Audio Processing Block DSP Assembly Code

3.1.3 Output of Processed Audio Data

The TAS3xxx architecture supports both digital and analog outputs. Immediately after the completion of the programmable audio processing blocks, the audio is available via the outputs(Registers DO1-DO8). Although there are no constraints on which Output Register are used for the output data variables, a programmer may find is useful to use all the available output registers for more efficient code. Table 3-2 shows an example of optimized use of the Input and Output Data Registers.

Note: Refer to the respective TAS3xxx Data Manuals for exact input and output options available

ALU1	ALU2	Data RAM Load to Register	COEF RAM Load to Register	RAM Store
NOP |	CLRACC |	LD(SDOUT1L_D,L) |	NOP |	NOP
THRU(L) |	NOP |	LD(SDOUT1R_D,L) |	NOP |	ST(DI,DATA,SDIN1L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO1) |	LD(SDOUT2L_D,L) |	NOP |	ST(DI,DATA,SDIN1R_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO2) |	LD(SDOUT2R_D,L) |	NOP |	ST(DI,DATA,SDIN2L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO3) |	LD(SDOUT3L_D,L) |	NOP |	ST(DI,DATA,SDIN2R_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO4) |	LD(SDOUT3R_D,L) |	NOP |	ST(DI,DATA,SDIN3L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO5) |	LD(SDOUT4L_D,L) |	NOP |	ST(DI,DATA,SDIN3R_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO6) |	LD(SDOUT4R_D,L) |	NOP |	ST(DI,DATA,SDIN4L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO7) |	NOP |	NOP |	ST(DI,DATA,SDIN4R_D)
NOP |	ADD(ACC,LR,CLP32,DO8) |	NOP |	NOP |	NOP

Table 3-2: Optimized DSP Assembly Code

3.1.4 Real-Time Control of Audio Processing

Real-Time control of audio processing is provided by a hardware based I^2C interface of the TAS3xxx device. The I^2C interface is facilitated by a TAS3xxx hardware block that interfaces with the embedded M8051 Microcontroller. This communication is accomplished though the I^2C slave interface in which the external system controller is the master I^2C device and the TAS3xxx is the I^2C slave device. Legal Master and Slave Addresses are available in the respective TAS3xxx Data Manuals.

The smallest I^2C transaction in a command is four bytes. The M8051 Microcontroller is (by default) programmed so that the maximum number of bytes in an I^2C command is 20 bytes. This facilitates sending five filter coefficients at a time since one of the main audio components is an IIR Biquad filter that requires five filter coefficients (four bytes per coefficient). In the case of the four input mixer

component shown in the example above, the four gain values (GAIN1, GAIN2, GAIN3, and GAIN4) can be send from the system controller to the TAS3xxx Device with one I^2C write containing 16 bytes.

3.2 Advanced Audio Processing Features of the TAS3xxx DSP Core

The TAS3xxx DSP includes the following functions in the core

- Single Cycle 48 bit by 28 bit Multiply
- 76 bit Addition
- Multiply-Accumulate Capability
- Approximate Log_2 and Anti-Log_2 mathematical operations

These features are explained in the following sections

3.2.1 Single Cycle Multiply

There is no explicit multiply instruction in the TAS3xxx DSP instruction set. For this purpose, a 48 bit by 28 bit multiply operation is implicitly performed every DSP clock cycle and is controlled by the data loaded into registers MC and MD. The output of each multiply is stored in Register MR after two DSP clock cycles.

Note: If the contents of MC and MD are not changed, the MR register contains the same value.

3.2.2 76 Bit Addition

The 76 bit addition block has two input operands, A and B. Operand A can be from the 76 bit accumulator register or the 48 bit BR register. Operand B can be from the 48 bit LR register or from the 76 bit MR register or 76-bit ZERO register.

When a 48 bit data values is used as an operand, three automatic transformations done to convert the data into a 76 bit value.

- The original 48 bit adder data is input from either BR or LR.
- 28 zeros are concatenated following the Least Significant Bit of the 48 bit data
- The result of the concatenation is arithmetically shifted right by five bits, and signed extended

The following figure graphically shows the automatic transformation that occurs to each 48 bit data register.

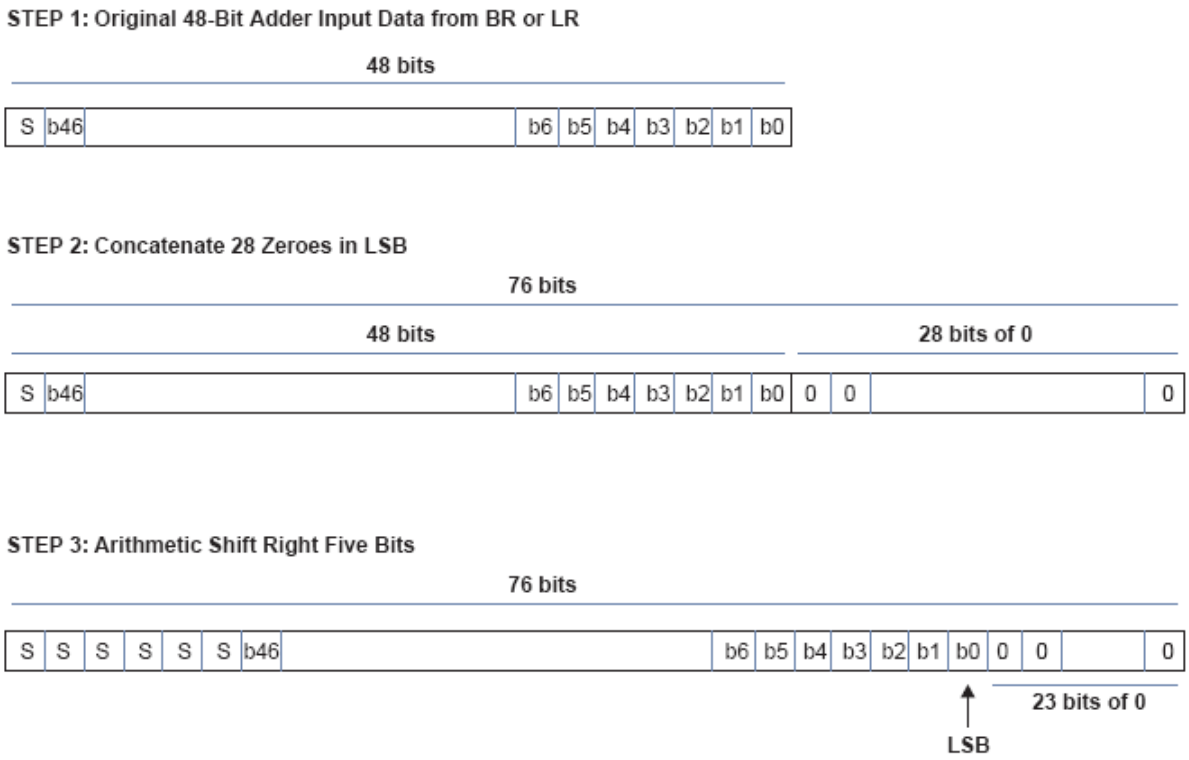


Figure 3-3: Automatic 48 bit to 76 bit conversion

The 76 bit result of the addition is then clipped from a 76 bit result into a 48 bit data value. This process is shown in Figure 3-6.

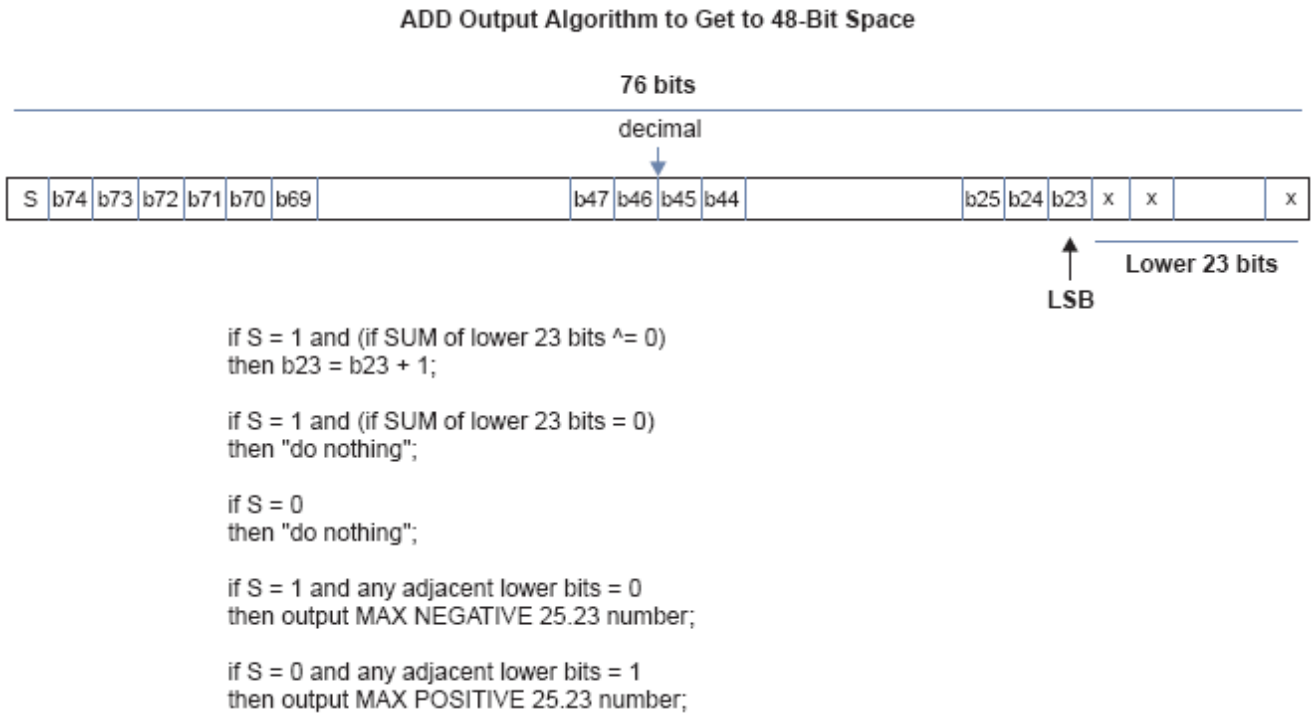


Figure 3-4: 76 bit Clip to a 48 bit data value

3.2.3 Multiply- Accumulate

Figure 3-7 shows how the developer can implement a multiply-accumulate operation using the TAS3xxx DSP assembly language.

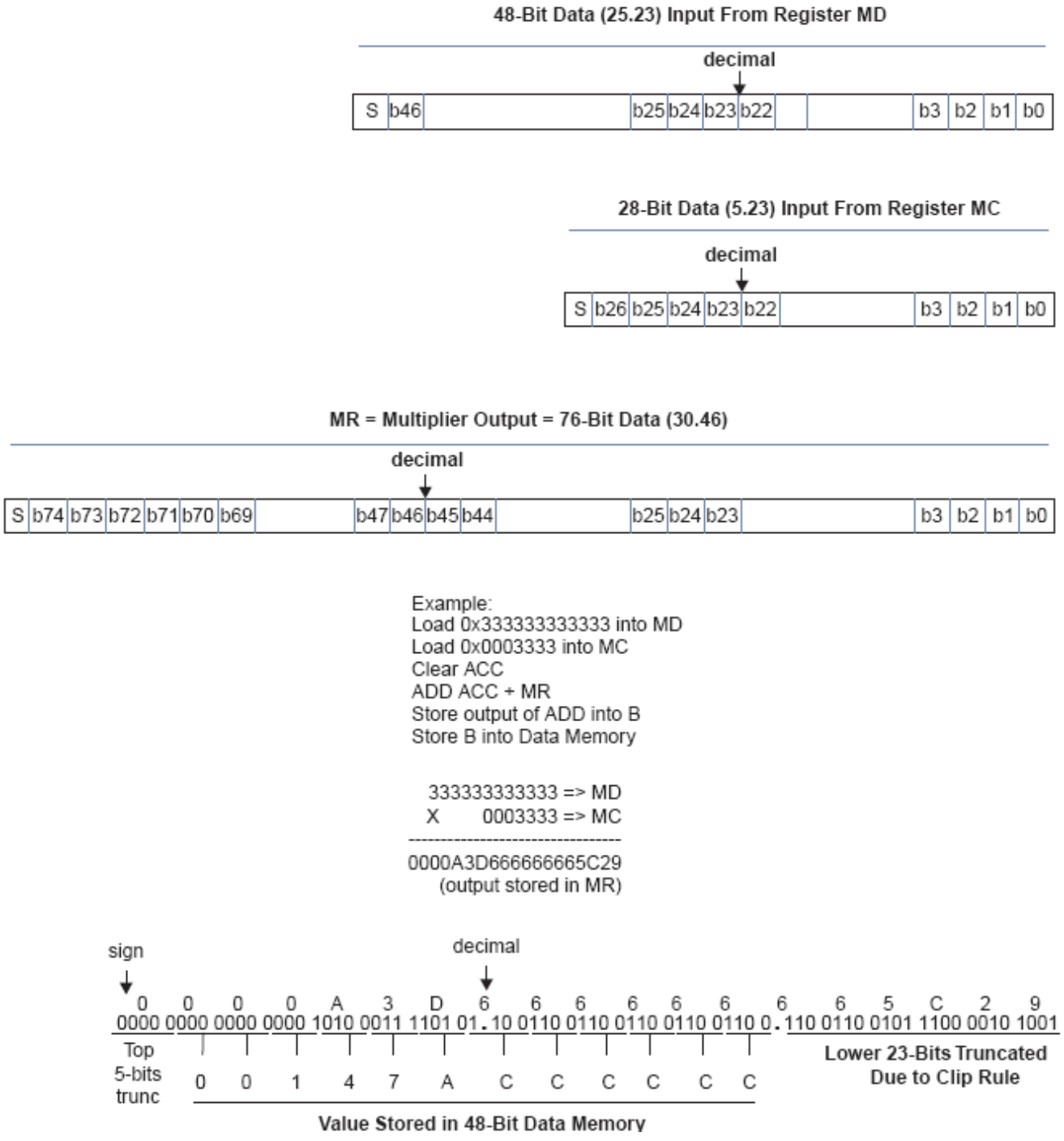


Figure 3-5: Multiply-Accumulate implementation via TAS3xxx DSP Assembly Code

3.2.4 *Log$_2$ and Anti-Log$_2$ Operations*

Log$_2$ and Anti-Log$_2$ operations are normally used the scale data so that processing can be performed without clipping. The following figure shows how these operations are performed in the TAS3xxx DSP Core

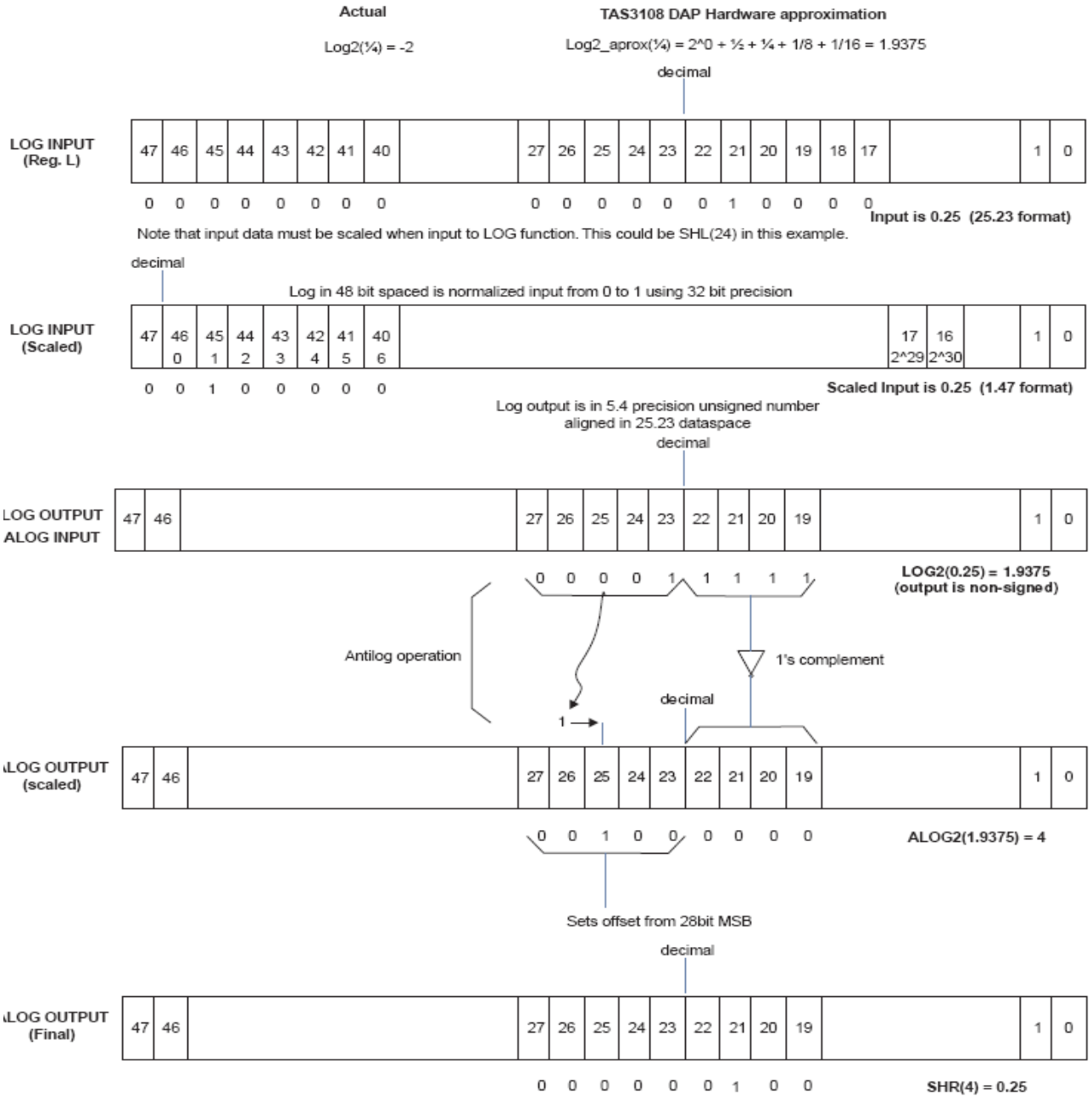


Figure 3-6:TAS3xxx DSP Log$_2$ and Anti-Log$_2$ Implementation

Chapter 4

DSP Instruction Set

4.1 DSP Instruction Word Format

The TAS3xxx Family of Digital Audio Processors uses a 54 bit instruction word which can simultaneously load two operands from Data RAM and Coefficient RAM, store the result, and perform two parallel arithmetic operations. Figure 4-1 shows the format of the 54 bit DSP Instruction word.

The instruction word is broken down into five sections. ALU Stage 1, ALU Stage 2, Memory Operation Stage 1, Memory Operation Stage 2, and Memory Operation Stage 3

ALU Stage 1 executes mathematical features that transform the contents of the B and L registers into the desired values that are send to the ALU.

ALU Stage 2 executes the arithmetic logic instructions to the specified input operands and returns the result.

Memory Operation Stage 1 loads the contents of the selected Data RAM Register into the specified register.

Memory Operation Stage 2 loads the contents of the selected Coefficient RAM Register into the specified register.

Memory Operation Stage 3 stores the contents of the specified register back into Data or Coefficient RAM, or into Register B, L, or MD.

ALU Stage 1	ALU Stage 2	Memory Operation Stage 1	Memory Operation Stage 2	Memory Operation Stage 3
53 49	48 42	(Data RAM Loads) 41 27	(Coefficient RAM Loads) 26 14	(Data & Coefficient RAM Stores) 13 0

Figure 4-1: TAS3xxx DSP Instruction Word Format

4.2 DSP Assembly Language Instruction Set

The following tables outline the DSP Assembly Instructions for the TAS3xxx Family of Devices for each stage of the Digital Signal Processor core. Details about each of the instructions are available in the sections that follow.

4.2.1 ALU Stage 1 Instruction Set

Table 4-1: ALU Stage 1 Instruction Set

Instruction	Function	Functional Description
ABS (reg)	Absolute Value	Performs Absolute value on data in (*reg)*
NEG (reg)	Performs 2's Complement	Negates data in (*reg)*
LOG2	Base Two Logarithm	Performs Base Two Log of data in (*reg*)
ALOG2	Base Two Anti-Logarithm	Performs Base Two Anti-Log of data in (reg)
SHR (bits)	Logical Shift Right	Shifts data in Register B right by (*bits) bits*
SHL (bits)	Shift Left	Shifts data in Register B left by (*bits*) bits
CLRACC	Clear Accumulator	Clears Accumulator Register
THRU (reg)	Transfer Register to next stage	Copies Register B to BR, and Register L to LR
JMP	Absolute Jump	Used in conjunction with PCADDR
BOC	Branch when A Comp to B = 1	Used in conjunction with PCADDR
BNC	Branch when A Comp to B = 0	Used in conjunction with PCADDR
STOP	Stop Program Counter	PC
Label		Parameter of PCADDR: PCADDR(*Label)*
NOP	No Operation	-

4.2.2 ALU Stage 2 Instruction Set

Table 4-2: ALU Stage 2 Instruction Set

Instruction	Function	Functional Description
CLRACC	Clear Accumulator	Clears Accumulator Register
ADD (a,b,c,d)	Add operands a and b w/clip c, store in reg d	Adds a to b and stores in register d with clip parameter c
COMP (reg1, reg2)	Compare registers 1 and 2	Compares operands [ACC or BR] to [LR or MR]
NOP	No Operation	-

4.2.3 Memory Operations Stage 1 Instructions

Table 4-3: Memory Operation Stage 1 Instruction Set

Instruction	Function	Functional Description
LD (addr, reg)	Load	Loads value from Data Ram at *addr to* reg
LDC (addr, reg)	Load when A Comp to B = 1	Loads value from Data Ram at *addr to* reg when compare = 1
LNC (addr, reg)	Load when A Comp to B = 0	Loads value from Data Ram at *addr to* reg when compare = 0
NOP	No Operation	-

4.2.4 Memory Operations Stage 2 Instructions

Table 4-4: Memory Operation Stage 2 Instruction Set

Instruction	Function	Functional Description
LD (addr, reg)	Load	Loads value from Coefficient Ram at *addr to* reg
LDC (addr, reg)	Load when A Comp to B = 1	Loads value from Coef Ram at *addr to* reg when compare = 1
LNC (addr, reg)	Load when A Comp to B = 0	Loads value from Coef Ram at *addr to* reg when compare = 0
NOP	No Operation	-

4.2.5 Memory Operations Stage 3 Instructions

Table 4-5: Memory Operation Stage 3 Instruction Set

Instruction	Function	Functional Description
ST (reg, DATA, addr)	Store register to Data RAM	Stores value in *reg* to Data RAM at address *addr*
ST (reg, COEF, addr)	Store register to Coefficient RAM	Stores value in reg to coef RAM at address addr
PCADDR	Assign Label	Used in conjunction with JMP, BOC, or BNC
NOP	No Operation	-

4.3 Detailed Description of TAS3xxx DSP Instruction Set

4.3.1 ABS (reg)

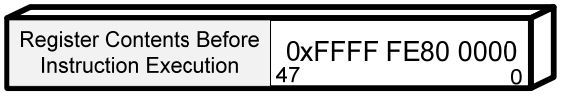
Description:

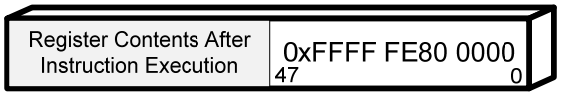
This instruction computes the absolute value of the data in Register B or Register L. The result of this instruction is available in the BR or LR register on the start of the next DSP clock cycle.

Syntax:	ABS(B)	**Opcode:**	00001
	ABS(L)		01110

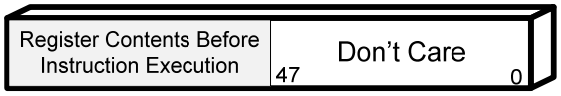
Example 1: **ABS(B)** | NOP | NOP | NOP | NOP

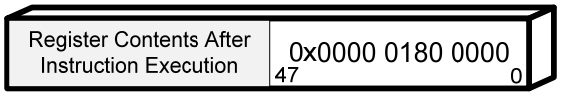
Register B:





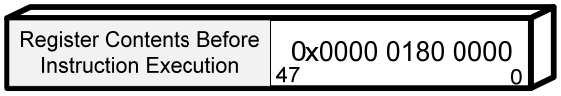
Register BR

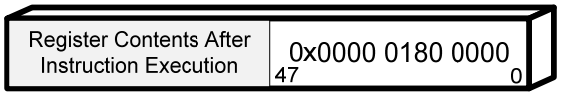




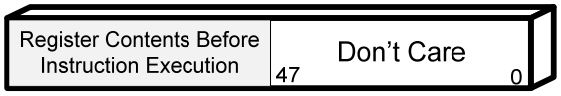
Example 2: **ABS(L)** | NOP | NOP | NOP | NOP

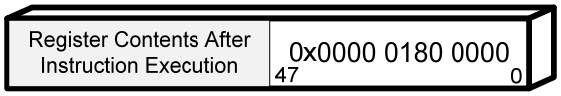
Register L:





Register LR





4.3.2 ADD (rega,regb,para, regd)

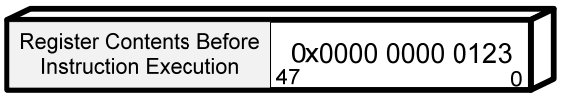
Description:

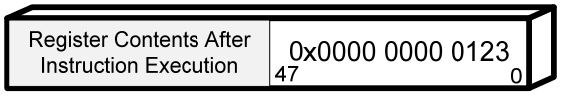
This instruction adds Register *rega* to Register *regb*. The 76 result of addition is clipped by the parameter *para* and is then stored is register *regd.* Register *rega* can be the ACC or BR. Register *regb* can be Register (LR, MR, or ZERO). *para* can be either no clip, a 32 bit clip, or a 28 bit clip (NONE, CLIP32, or CLIP28). Register *regd* can be Regsiter (ACC, B, DLYI, L, MC, MD or DO1, DO2, ….DO8)

Syntax:	**Opcode:**
ADD(ACC,LR,CLP32,DO1)	011 1001
ADD(ACC,LR,CLP32,DO2)	011 1010
ADD(ACC,LR,CLP32,DO3)	011 1011
ADD(ACC,LR,CLP32,DO4)	011 1100
ADD(ACC,LR,CLP32,DO5)	110 1111
ADD(ACC,LR,CLP32,DO6)	110 0010
ADD(ACC,LR,CLP32,DO7)	110 0011
ADD(ACC,LR,CLP32,DO8)	111 0011
ADD(ACC,ZERO,NONE,B)	101 0000
ADD(BR,LR,CLP32,DO1)	011 0001
ADD(BR,LR,CLP32,DO2)	011 0010
ADD(BR,LR,CLP32,DO3)	011 0011
ADD(BR,LR,CLP32,DO4)	011 0100
ADD(BR,LR,CLP32,DO5)	110 1101
ADD(BR,LR,CLP32,DO6)	101 1110
ADD(BR,LR,CLP32,DO7)	101 1111
ADD(BR,LR,CLP32,DO8)	111 0001
ADD(BR,LR,NONE,B)	000 1010
ADD(BR,LR,NONE,L)	000 1011
ADD(BR,LR,NONE,MC)	000 1100
ADD(BR,LR,NONE,MD)	000 1001
ADD(BR,LR,NONE,DLYI)	000 1111
ADD(BR,MR,CLP32,DO1)	010 1101
ADD(BR,MR,CLP32,DO2)	010 1110
ADD(BR,MR,CLP32,DO3)	010 1111
ADD(BR,MR,CLP32,DO4)	010 1000
ADD(BR,MR,CLP32,DO5)	110 1100
ADD(BR,MR,CLP32,DO6)	101 1100
ADD(BR,MR,CLP32,DO7)	101 1101
ADD(BR,MR,CLP32,DO8)	111 0000
ADD(BR,MR,NONE,ACC)	000 0101
ADD(BR,MR,NONE,B)	000 0010
ADD(BR,MR,NONE,DLYI)	000 0111
ADD(BR,MR,NONE,L)	000 0011
ADD(BR,MR,NONE,MC)	000 0100
ADD(BR,MR,NONE,MD)	000 0001
ADD(BR,ZERO,CLP32,B)	101 1011
ADD(BR,ZERO,NONE,B)	101 1000
ADD(BR,ZERO,NONE,L)	101 1001
ADD(BR,ZERO,NONE,MC)	101 1010
ADD(BR,ZERO,NONE,MD)	101 0111

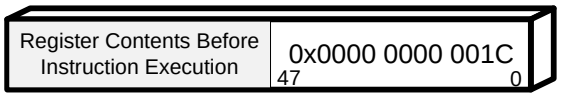
Example 1: NOP | **ADD(BR,MR,CLP32,DO3)** | NOP | NOP | NOP

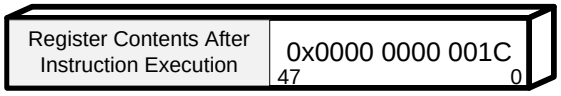
Register MR:



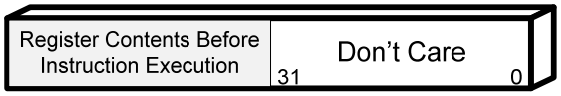


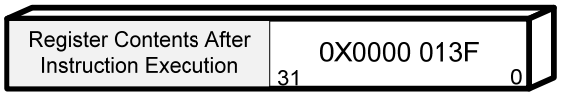
Register BR





Register DO3:





4.3.3 ALOG2

Description:

This instruction computes the approximate inverse Base 2 Logarithm of the data in Register L. The result of this instruction is available in the Register LR on the start of the next DSP clock cycle.

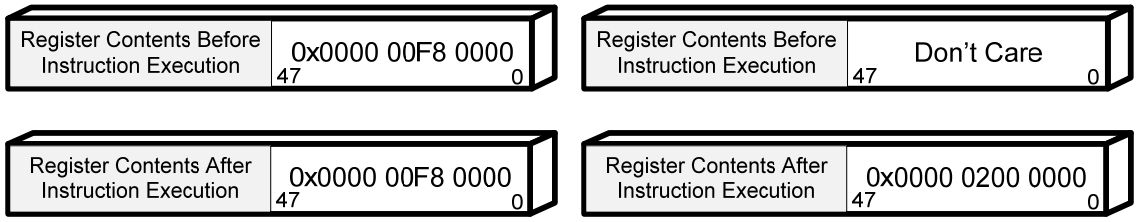
The input is in base 2 log space (5.4 precision unsigned number) with a range of 0.0000 to 31.9375. The output is in linear space (25.23 precision unsigned number)

Note: The base 2 anti-logarithm is only available for the contents of Register L

Syntax: ALOG2(L) **Opcode:** 00100

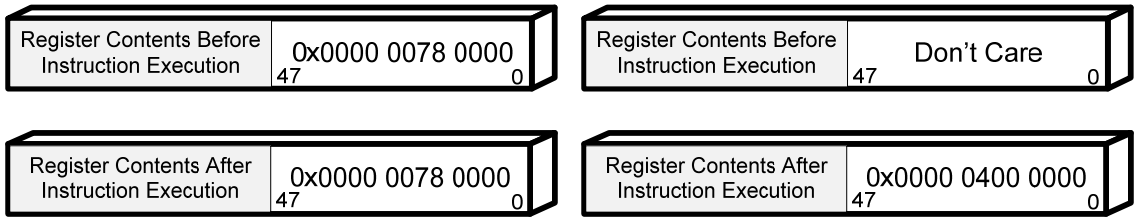
Example 1: **ALOG2(L)** | NOP | NOP | NOP | NOP

Register L: **Register LR**



Example 2: **ALOG2(L)** | NOP | NOP | NOP | NOP

Register L: **Register LR**



4.3.4 BOC

Description:

This instruction executes a conditional branch to *label* if the value of operand A is equal to the value of operand B. This results when a compare instruction of these operands results in a logic 0.

The PCADDR instruction is always used with the BOC instruction

Note: When a BOC is performed, no other instructions (Except NOPs) are allowed on that cycle

Syntax: BOC **Opcode:** 10011

Example 1: **BOC** | NOP | NOP | NOP | PCADDR(BRANCH_PT)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

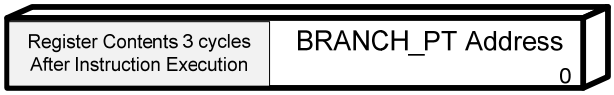
NOP | NOP | NOP | NOP | NOP

< other processing(optional) >

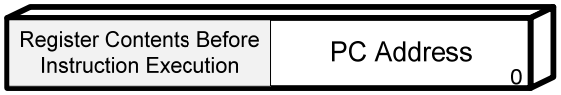
BRANCH_PT | NOP | NOP | NOP | NOP

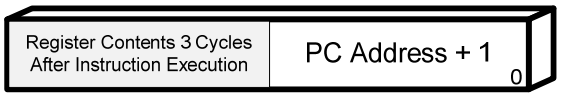
Register PC:

If Operand A = Operand B



If Operand A ≠ Operand B





4.3.5 BNC

Description:

This instruction executes a conditional branch to *label* if the value of operand A is not equal to the value of operand B. This results when a compare instruction of these operands results in a logic 1.

Syntax: BNC **Opcode:** 10100

Example 1: **BNC** | NOP | NOP | NOP | PCADDR(BRANCH_PT)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

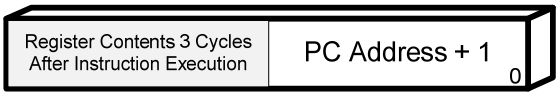
NOP | NOP | NOP | NOP | NOP

< other processing(optional) >

BRANCH_PT| NOP | NOP | NOP | NOP

Register PC:

If Operand A = Operand B



If Operand A ≠ Operand B

4.3.6 CLRACC

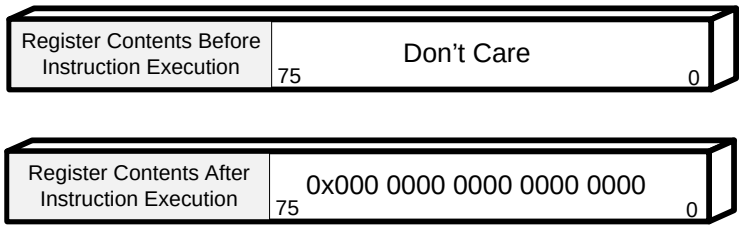
Description:

This instruction clears the contents of the 76 bit accumulator register. On the next DSP clock cycle, the Accumulator Register will contain all zeros.

Syntax: CLRACC **Opcode:** 01111

Example 1: **CLRACC** | NOP | NOP | NOP | NOP

Register ACC:



4.3.7 COMP (rega, regb)

Description:

This instruction compares the sign of Register *rega* with the sign of Register *regb* where *rega* is the accumulator or Register BR, and *regb* may be either , Register LR, Register MR or Register ZERO. The result of the compare function is input to a multiplexer that determines the compare result. See the figure 4-2 for details.

Note: The result of the COMP Instruction will be available two DSP clock cycles after the instruction is executed.

There are six cases that correspond to the compare instruction:

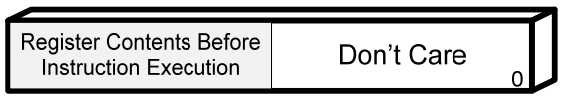
1. If Operand A and Operand B are positive, then the result of the compare is 0
2. If Operand A and Operand B are negative, then the result of the compare is 1
3. If Operand A is negative and Operand B is positive and the absolute value of both Operands are equal, then the result of the compare is 0
4. If Operand A is positive and Operand B is negative and the absolute value of both Operands are equal, then the result of the compare is 1
5. If the absolute value of Operand A is greater than the absolute value of Operand B, then the result of the compare is 1
6. If the absolute value of Operand A is less than the absolute value of Operand B, then the result of the compare is 0.

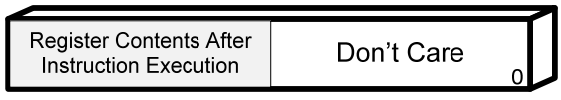
A table of test values and the result is shown below in Table 4-6.

Syntax:	COMP(ACC, LR)	**Opcode:**	010 0000
	COMP(ACC, MR)		001 10000
	COMP(BR, LR)		001 0000
	COMP(BR, MR		000 1000

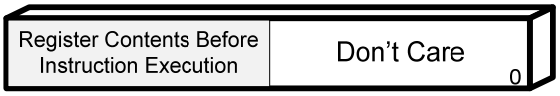
Example 1: NOP | **COMP(ACC, LR)** | NOP | NOP | NOP

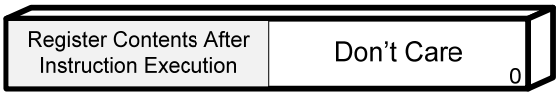
Register ACC:





Register LR





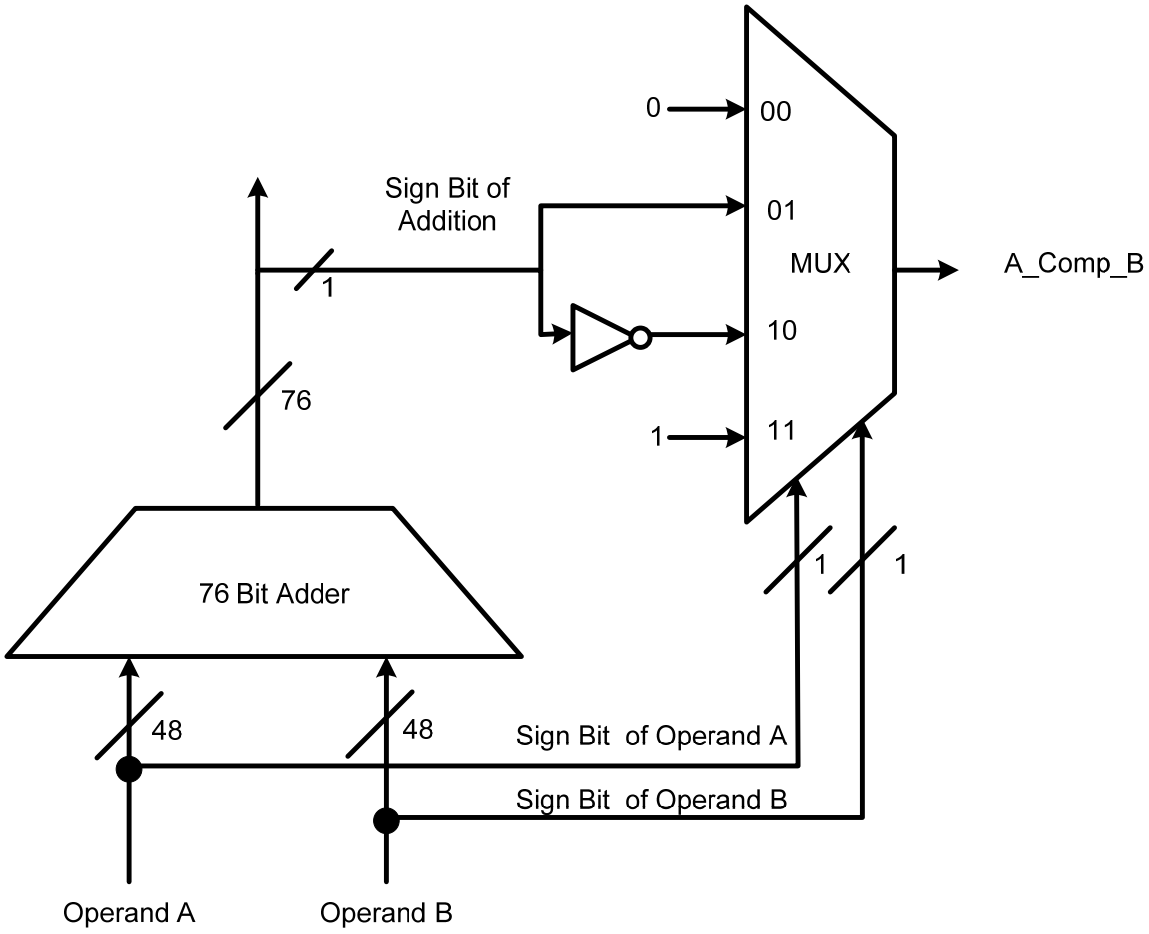


Figure 4-2: TAS3xxx Comparison Hardware

Table 4-6: Test Values for TAS3xxx Comparison Hardware.

Operand_A	Operand_B	sum	A_CP_B	Rule
1	1	2	0	1
-1	1	0	0	3
1	-1	0	1	4
-1	-1	-2	1	2
2	1	3	0	1
-2	1	-1	1	5
2	-1	1	1	5
-2	-1	-3	1	2
1	2	3	0	1
-1	2	1	0	6
1	-2	-1	0	6
-1	-2	3	1	2
0	1	1	0	1
1	0	1	0	1
0	-1	-1	0	6
-1	0	-1	1	5
0	0	0	0	1

4.3.8 DLYPTR(ptr)

Description:

This instruction is used to indicate which 10 bit delay memory pointer is used for the current audio delay function.

Syntax: DLYPTR(ptr) **Opcode:** 0000

Example 1: NOP | CLRACC | LD(DelayIn1_D,L, NONE) | NOP | NOP

THRU(L) | NOP | NOP) | NOP | NOP

NOP | ADD(ACC,LR,NONE,DLYI) | NOP | NOP | **DLYPTR(0)**

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | ST(DLYO,DATA,DelayOut1_D)

4.3.9 JMP

Description:

This instruction executes an unconditional program counter jump to the address specified by *label*

The PCADDR instruction is always used with the JMP instruction. The label can be anywhere in the code outside the four lines of code specified for the JMP instruction

Note 1: The JMP instruction must be followed by three DSP Clock Cycles of NOPs. The three DSP clock cycles of NOPs allow enough time to fully clear the instruction pipe.

Note 2: When the JMP instruction is performed, no other instructions (except NOPs) may be executed on that cycle

Note 3: Due to HW limitations is up to the developer to guarantee that all JMP instruction occur within the first 1K for the TAS3108 and TAS3204, and within the first 2K for TAS3208, TAS3308.

Syntax: JMP **Opcode:** 10010

Example 1: **JMP** | NOP | NOP | NOP | PCADDR(PMRK_A)

NOP | NOP | NOP | NOP | NOP

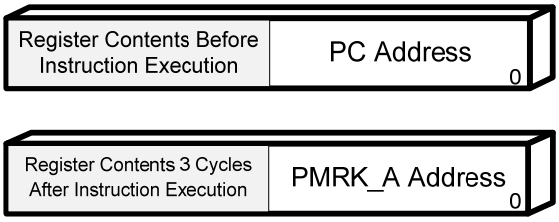
NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

< other processing(optional) >

PRMK_A| NOP | NOP | NOP | NOP

Register PC:



4.3.10 LD(mem_addr, reg)

Description:

This instruction executes an unconditional load of the contents of memory address *mem_addr* into Register *reg*, where *reg* is Register B, L, or MD (for 48 bit Data) or Register MC (for 28 bit Coefficient). mem_addr is10 bit Data RAM or Coefficient RAM address of the desired register to be loaded.

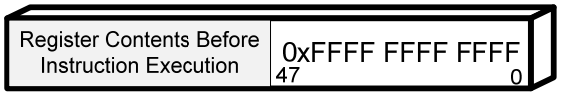
Note: Data RAM loads belong to Memory Operation Stage 1 Instruction Set, where as Coefficient RAM loads belong to Memory Operation Stage 2 Instruction Set.

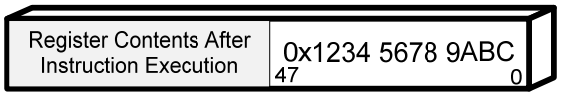
Important: Specifying Data RAM loads in Memory Operation Stage 2, or Coefficient loads in Memory Operation Stage 1 is Illegal.

Syntax:	LD(mem_addr,B)	**Opcode:**	0 0010
	LD(mem_addr, L)		0 0011
	LD(mem_addr, BL)		1 1100
	LD(mem_addr, MC)		1 1111
	LD(mem_addr, MD)		0 0001

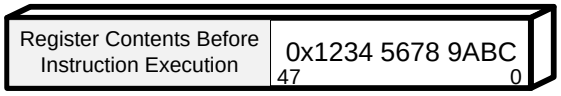
Example 1: NOP | NOP | **LD(mem_addr, B)** | NOP | NOP

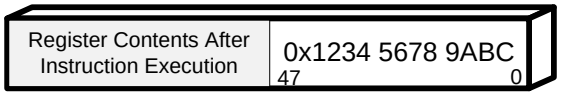
Register B:





Register mem_addr:





4.3.11 LDC(mem_addr, reg)

Description:

This instruction executes a conditional load of contents of data memory address mem_addr into Register *reg* when the compare instruction results in a logic 1.

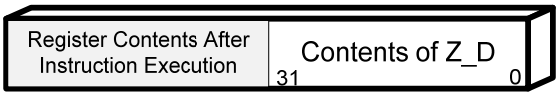
Register *reg* can be Register B, L, or MD for 48 bit data or Register MC for 28 bit data.

Syntax:	LDC(mem_addr,B)	**Opcode:**	0 0101
	LDC(mem_addr, L)		0 0110
	LDC(mem_addr, MD)		1 0100

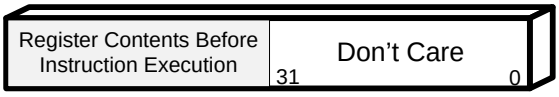
Example 1:
NOP | NOP | LD(X_D, B) | NOP | NOP
NEG(B) | NOP | LD(ZERO_D, L) | NOP | NOP
THRU(L) | NOP | NOP | NOP | NOP
NOP | COMP(BR,LR)| NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | LNC(Y_D, MD) | NOP | NOP
NOP | NOP | **LDC(Z_D, MD)** | NOP | NOP

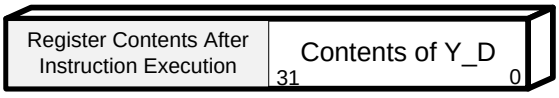
Register MD:

If content of BR = Content of LR



If Operand A ≠ Operand B





4.3.12 LNC(D_addr, reg)

Description:

This instruction executes a conditional load of contents of data memory address mem_addr into Register *reg* when the compare instruction results in a logic 0.

Register *reg* can be Register B, L, or MD for 48 bit data or Register MC for 28 bit data.

Note: Data RAM loads belong to Memory Operation Stage 1 Instruction Set, where as Coefficient RAM loads belong to Memory Operation Stage 2 Instruction Set.

Important: Specifying Data RAM loads in Memory Operation Stage 2, or Coefficient loads in Memory Operation Stage 1 is Illegal.

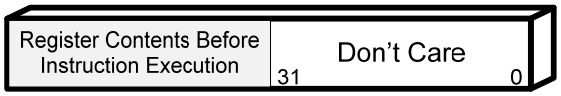
Syntax:	LNC(D_addr,B)	**Opcode:**	0 1000
	LNC(D_addr, L)		0 1001
	LNC(D_addr, MD)		1 0111

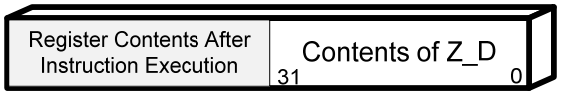
Example 1:

NOP | NOP | LD(X_D, B) | NOP | NOP
NEG(B) | NOP | LD(ZERO_D, L) | NOP | NOP
THRU(L) | NOP | NOP | NOP | NOP
NOP | COMP(BR,LR)| NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | **LNC(Y_D, MD**) | NOP | NOP
NOP | NOP | LDC(Z_D, MD) | NOP | NOP

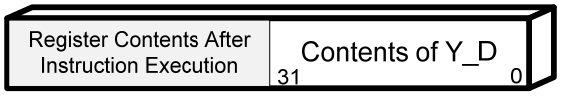
Register MD:

If content of BR = Content of LR





If content BR ≠ Content of LR



4.3.13 LOG2

Description:

This instruction computes the approximate Base 2 Logarithm of the data in Register L. The result of this instruction is available in the Register LR on the start of the next DSP clock cycle.

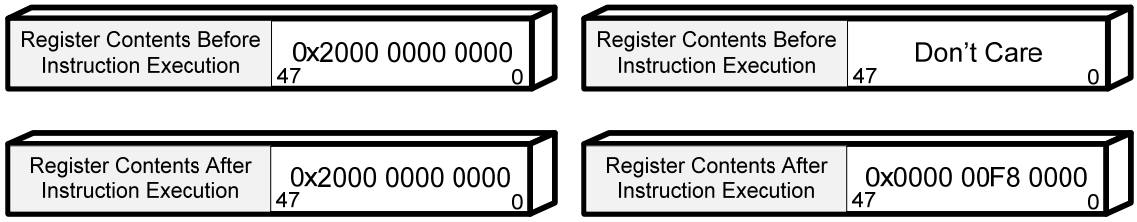
The input is in linear space (25.23 precision unsigned number) with a range of 0.0000 to approximately 16777216. The output is in base 2 log space (5.4 precision unsigned number)

Note: The base 2 logarithm is only available for the contents of Register L

Syntax: LOG2(L) **Opcode:** 00100

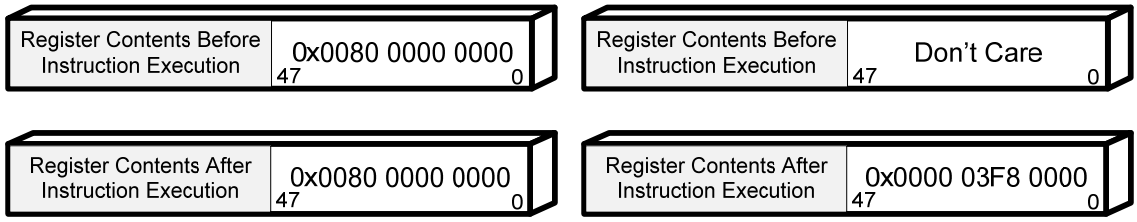
Example 1: **LOG2(L)** | NOP | NOP | NOP | NOP

Register L: **Register LR**



Example 2: **LOG2(L)** | NOP | NOP | NOP | NOP

Register L: **Register LR**



4.3.14 NEG (reg)

Description:

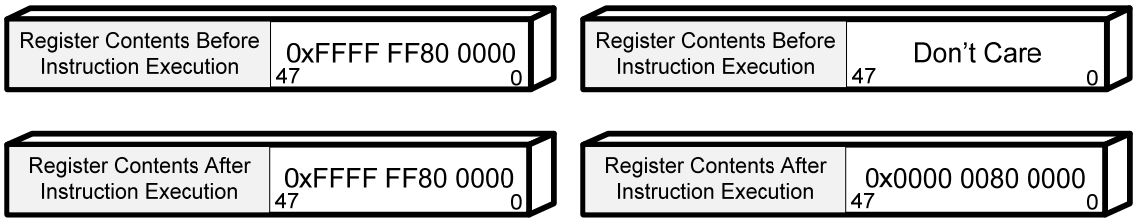
This instruction computes the 2's Complement on the value of the data in Register B or Register L. The result of this instruction is available in the BR or LR register on the start of the next DSP clock cycle.

Syntax: NEG(B) **Opcode:** 00010

NEG(L) 00011

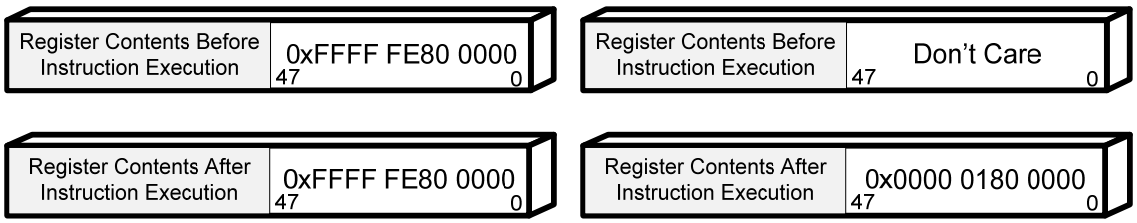
Example 1: **NEG(B)** | NOP | NOP | NOP | NOP

Register B: **Register BR**



Example 2: **NEG(L)** | NOP | NOP | NOP | NOP

Register L: **Register LR**



4.3.15 NOP

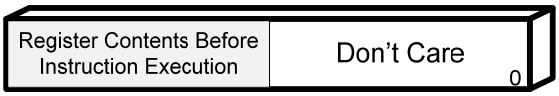
Description:

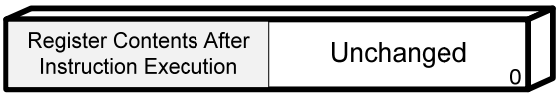
This instruction does not execute any operation

Syntax: NOP **Opcode:** 00000

Example 1: **NOP** | **NOP** | **NOP** | **NOP** | **NOP**

All Registers:





4.3.16 SHL (bits)

Description:

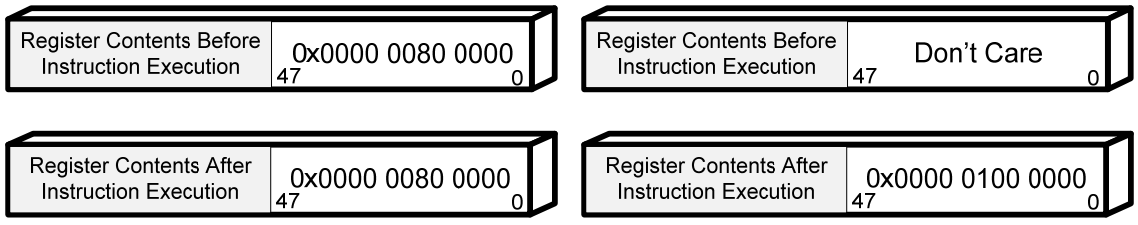
This instruction shifts the contents of Register B by the number of bits specified. The result is available in Register BR at the start of the next DSP clock cycle.

Note: Shift right operations are only available for the contents of Register B.

Syntax:	SHL (1)	**Opcode:**	01010
	SHL (2)		00111
	SHL (3)		01100
	SHL (4)		01101
	SHL (20)		10110

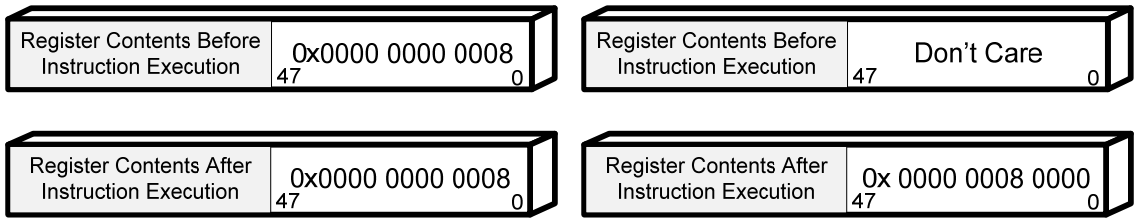
Example 1: **SHL (1)** | NOP | NOP | NOP | NOP

Register B: **Register BR**



Example 2: **SHL (20)** | NOP | NOP | NOP | NOP

Register B: **Register BR**



4.3.17 SHR (bits)

Description:

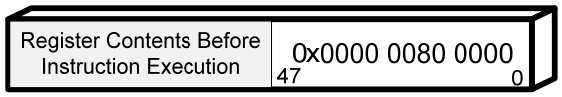
This instruction shifts the contents of Register B by the number of bits specified. The result is available in Register BR at the start of the next DSP clock cycle.

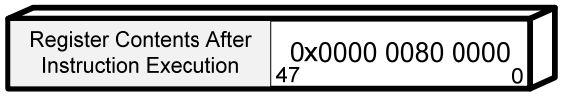
Note: Shift right operations are only available for the contents of Register B.

Syntax:	SHR(1)	**Opcode:**	00100
	SHR(2)		00111
	SHR(3)		01000
	SHR(4)		01001

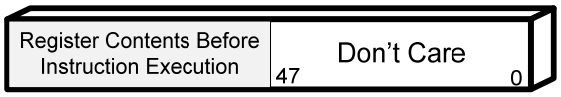
Example 1: **SHR (3)** | NOP | NOP | NOP | NOP

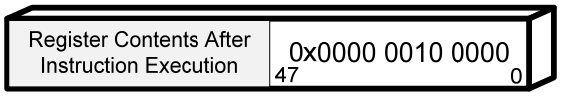
Register B:





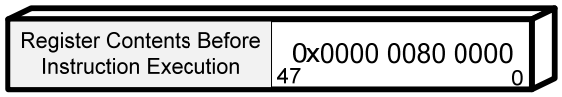
Register BR

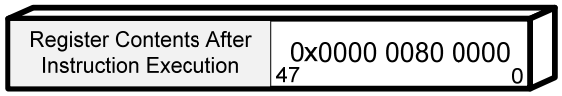




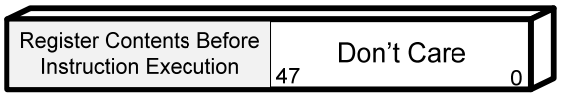
Example 2: **SHR (4)** | NOP | NOP | NOP | NOP

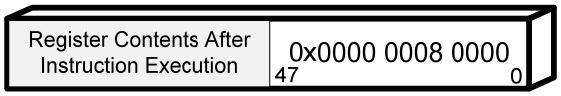
Register B:





Register BR





*4.3.18 ST(*reg, COEF, mem_addr)

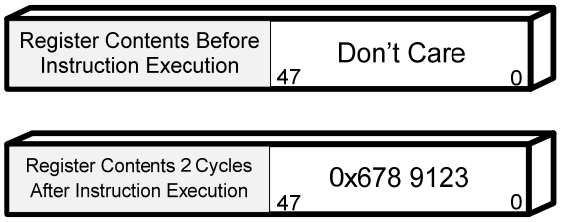
Description:

This instruction stores the lower 28 bits contents of Register L, B, or MD into the 10 bit coefficient RAM address mem_addr.

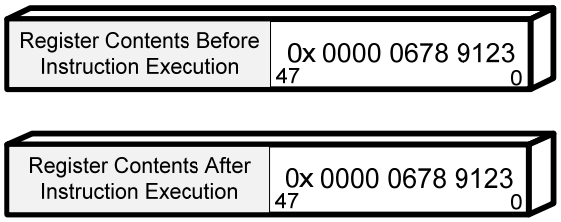
Syntax:	ST(L,COEF,C_addr)	**Opcode:**	0110
	ST(L, COEF, C_addr)		0111
	ST(B, COEF, C_addr)		1000

Example 1: NOP | NOP | NOP | NOP | **ST(L, COEF, VAR1_C)**

Data Ram Register VAR1_C:



Register L



*4.3.19 ST(*DI, DATA, mem_addr)

Description:

This instruction stores the contents of Register DI (Digital Audio Data Input) into the 10 bit data RAM address mem_addr.

Register DI is a special register used to access audio data received by the DSP. The ST(DI, DATA,D_addr) instruction copies data from DI into Data RAM for later processing

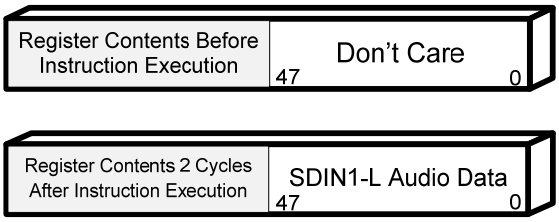
Note: The TAS3xxx least three least significant bits of the data RAM address indicate which input channel is copied to memory.

Syntax: ST(DI,DATA,D_addr) **Opcode:** 0110

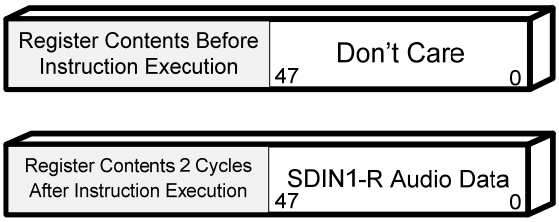
Example 1: NOP | NOP | NOP | NOP | **ST(DI, DATA, SDIN1L_D)**

NOP | NOP | NOP | NOP | **ST(DI, DATA, SDIN1R_D)**

Data RAM Register SDIN1L_D:



Data RAM Register SDIN1R_D



*4.3.20 ST(*DLYO, DATA, mem_addr)

Description:

This instruction stores the contents of Register DLYO (Delay Memory Out) into the 10 bit data memory address mem_addr.

Register DLYO is a special register used to access delayed audio data samples from the delay memory. There must be 13 cycles between the delay pointer instruction DLYPTR and the ST instruction. These 13 cycles can either contain NOPs or other processing if required.

Syntax: ST(DLYO,DATA,D_addr) **Opcode:** 0100

Example 1: NOP | CLRACC | LD(DelayIn1_D,L,NONE) | NOP | NOP

THRU(L) | NOP | NOP | NOP | NOP

NOP | NOP | ADD(ACC, LR, NONE, DELYI) | NOP | DLYPTR(0)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

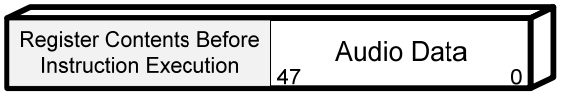
NOP | NOP | NOP | NOP | NOP

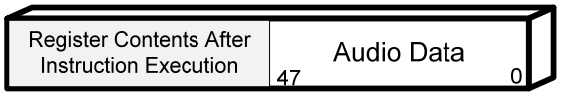
NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

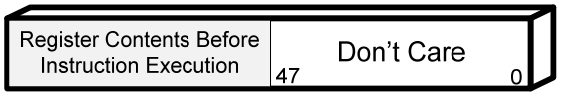
NOP | NOP | NOP | NOP | **ST(DLYO,DATA,DelayOut1_D)**

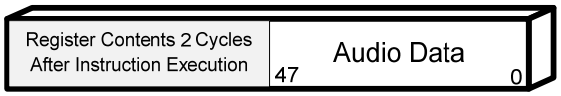
Register DLYO:





Data RAM Register DelayOut1_D





*4.3.21 ST(*reg, DATA, mem_addr)

Description:

This instruction stores the contents of Register *reg* into Data memory address mem_addr.

Register *reg* is Register B, L, or MD. *mem_addr* is a 10 bit Data RAM address

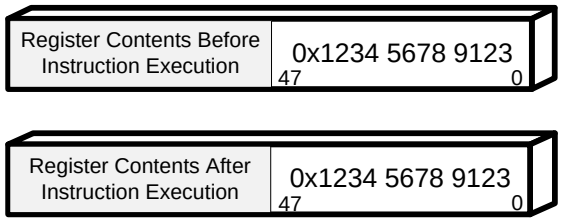
Note: Data RAM loads belong to Memory Operation Stage 1 Instruction Set, where as Coefficient RAM loads belong to Memory Operation Stage 2 Instruction Set.

Important: Specifying Data RAM loads in Memory Operation Stage 2, or Coefficient loads in Memory Operation Stage 1 is Illegal.

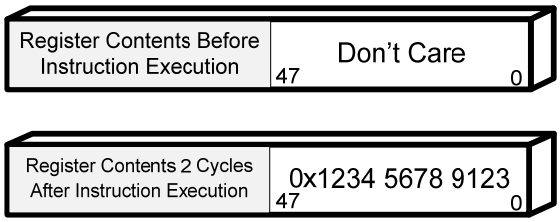
Syntax:	ST(L,DATA,mem_addr)	**Opcode:**	0011
	ST(MD,DATA,mem_addr)		0001
	ST(B,DATA,mem_addr)		0010

Example 1: NOP | NOP | NOP | NOP | **ST(B, DATA, VAR1_D)**

Register B:



Register VAR1_D



4.3.22 STOP

Description:

This instruction halts the Program Counter.

Important: Every DSP assembly program must have at least one STOP instruction (usually at the end of the program).

Note: If a Left/Right Clock error occurs at the same time as execution of the STOP instruction, the PC may miss the STOP instruction and erroneously continue. To guard against this, the following example shows a defensive programming technique to ensure that the Stop instruction is correctly executed.

Syntax: STOP **Opcode:** 10101

Example 1: BRANCH_PT | NOP | NOP | NOP | NOP

STOP | NOP | NOP | NOP | PCADDR(BRANCH_PT)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

JMP | NOP | NOP | NOP | PCADDDR(BRANCH_PT)

4.3.23 THRU

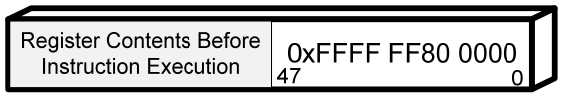
Description:

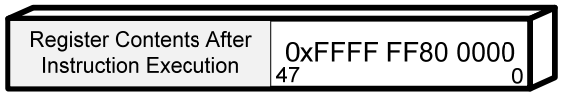
This instruction copies the contents of Register B to Register BR or the contents of Register L to Register LR

Syntax:	THRU (B)	**Opcode:**	10000
	THRU (L)		10001

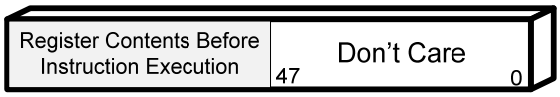
Example 1: **THRU (B)** | NOP | NOP | NOP | NOP

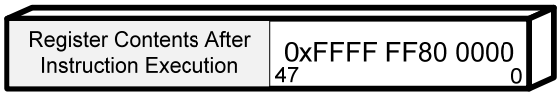
Register B:





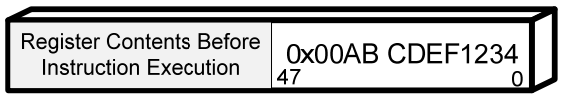
Register BR

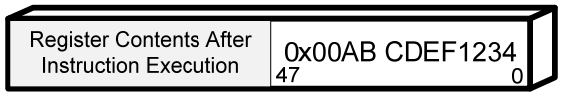




Example 2: **THRU (L)** | NOP | NOP | NOP | NOP

Register L:





Register LR