# TMS320C28x Floating Point Unit and Instruction Set

## **Reference Guide**



Literature Number: SPRUEO2A June 2007–Revised August 2008



## Contents

Prefa	ice					
1	Introduction7					
1.1	Introduction to the Central Processing Unit (CPU)					
1.2	Compatibility with the C28x Fixed-Point CPU					
	1.2.1 Floating-Point Code Development					
1.3	Components of the C28x plus Floating-Point CPU					
	1.3.1 Emulation Logic 10					
	1.3.2 Memory Map 10					
	1.3.3 On-Chip Program and Data 10					
	1.3.4 CPU Interrupt Vectors 10					
1.4	Memory Interface 10					
	1.4.1 Address and Data Buses 11					
	1.4.2 Alignment of 32-Bit Accesses to Even Addresses 11					
2	CPU Register Set 13					
2.1	CPU Registers 14					
	2.1.1 Floating-Point Status Register (STF) 16					
	2.1.2 Repeat Block Register (RB) 18					
3	Pipeline					
3.1	Pipeline Overview					
3.2	General Guidelines for Floating-Point Pipeline Alignment					
3.3	Moves from FPU Registers to C28x Registers					
3.4	Moves from C28x Registers to FPU Registers					
3.5	Parallel Instructions					
3.6	Invalid Delay Instructions					
3.7	Optimizing the Pipeline 27					
4	Instruction Set					
4.1	Instruction Descriptions					
4.2	Instructions 32					
Α	Revision History 137					
A.1	Changes 137					



#### List of Figures

1-1	FPU Functional Block Diagram	8
	C28x With Floating-Point Registers	
	Floating-point Unit Status Register (STF)	
2-3	Repeat Block Register (RB)	18
3-1	FPU Pipeline	22

#### List of Tables

2-1	28x Plus Floating-Point CPU Register Summary	15
	Floating-point Unit Status (STF) Register Field Descriptions	
2-3	Repeat Block (RB) Register Field Descriptions	18
4-1	Operand Nomenclature	30
4-2	Summary of Instructions	32
A-1	Technical Changes Made in This Revision	137



Preface SPRUE02A–June 2007–Revised August 2008

This document describes the CPU architecture, pipeline, instruction set, and interrupts of the C28x floating-point DSP.

#### About This Manual

The TMS320C2000<sup>™</sup> digital signal processor (DSP) platform is part of the TMS320<sup>™</sup> DSP family.

#### **Notational Conventions**

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h or with a leading 0x. For example, the following number is 40 hexadecimal (decimal 64): 40h or 0x40.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register.
     Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

#### **Related Documentation**

The following books describe the TMS320x28x and related support tools that are available on the TI website:

#### Data Manual and Errata—

SPRS439— <u>TMS320F28335</u>, <u>TMS320F28334</u>, <u>TMS320F28332</u>, <u>TMS320F28235</u>, <u>TMS320F28235</u>, <u>TMS320F28232</u>, <u>TMS320F28232</u> <u>Digital Signal Controllers (DSCs) Data Manual</u> contains the pinout, signal descriptions, as well as electrical and timing specifications for the F2833x/2823x devices.

SPRZ272— TMS320F28335, F28334, F28332, TMS320F28235, F28234, F28232 Digital Signal Controllers (DSCs) Silicon Errata describes the advisories and usage notes for different versions of silicon.

#### CPU User's Guides—

- **SPRU430** <u>TMS320C28x DSP CPU and Instruction Set Reference Guide</u> describes the central processing unit (CPU) and the assembly language instructions of the TMS320C28x fixed-point digital signal processors (DSPs). It also describes emulation features available on these DSPs.
- **SPRUEO2** <u>TMS320C28x Floating Point Unit and Instruction Set Reference Guide</u> describes the floating-point unit and includes the instructions for the FPU.

#### Peripheral Guides—

- **SPRU566** <u>TMS320x28xx, 28xxx Peripheral Reference Guide</u> describes the peripheral reference guides of the 28x digital signal processors (DSPs).
- **SPRUFB0** <u>TMS320x2833x</u>, <u>2823x</u> <u>System Control and Interrupts Reference Guide</u> describes the various interrupts and system control features of the 2833x digital signal controllers (DSCs).</u>
- **SPRU812** <u>TMS320x2833x</u>, <u>2823x</u> <u>Analog-to-Digital Converter (ADC)</u> <u>Reference Guide</u> describes how to configure and use the on-chip ADC module, which is a 12-bit pipelined ADC.</u>



Related Documentation

www.ti.com

- **SPRU949** <u>TMS320x2833x, 2823x External Interface (XINTF) User's Guide</u> describes the XINTF, which is a nonmultiplexed asynchronous bus, as it is used on the 2833x devices.
- SPRU963— <u>TMS320x2833x</u>, <u>TMS320x2823x</u> Boot ROM User's Guide describes the purpose and features of the bootloader (factory-programmed boot-loading software) and provides examples of code. It also describes other contents of the device on-chip boot ROM and identifies where all of the information is located within that memory.
- SPRUFB7— <u>TMS320x2833x, 2823x Multichannel Buffered Serial Port (McBSP) User's Guide</u> describes the McBSP available on the F2833x devices. The McBSPs allow direct interface between a DSP and other devices in a system.
- **SPRUFB8** <u>TMS320x2833x, 2823x Direct Memory Access (DMA) Reference Guide</u> describes the DMA on the 2833x devices.
- SPRUG04— <u>TMS320x2833x</u>, 2823x Enhanced Pulse Width Modulator (ePWM) Module Reference Guide describes the main areas of the enhanced pulse width modulator that include digital motor control, switch mode power supply control, UPS (uninterruptible power supplies), and other forms of power conversion.
- **SPRUG02** <u>TMS320x2833x</u>, <u>2823x High-Resolution Pulse Width Modulator (HRPWM)</u> describes the operation of the high-resolution extension to the pulse width modulator (HRPWM).
- **SPRUFG4** <u>TMS320x2833x</u>, <u>2823x</u> <u>Enhanced Capture (eCAP) Module Reference Guide</u> describes the enhanced capture module. It includes the module description and registers.</u>
- **SPRUG05** <u>TMS320x2833x</u>, <u>2823x</u> Enhanced Quadrature Encoder Pulse (eQEP) Reference Guide describes the eQEP module, which is used for interfacing with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine in high performance motion and position control systems. It includes the module description and registers.
- SPRUEU1— <u>TMS320x2833x</u>, <u>2823x</u> Enhanced Controller Area Network (eCAN) Reference Guide describes the eCAN that uses established protocol to communicate serially with other controllers in electrically noisy environments.
- SPRUFZ5— <u>TMS320F2833x, 2823x Serial Communication Interface (SCI) Reference Guide</u> describes the SCI, which is a two-wire asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format.
- SPRUEU3— TMS320x2833x, 2823x Serial Peripheral Interface (SPI) Reference Guide describes the SPI a high-speed synchronous serial input/output (I/O) port that allows a serial bit stream of programmed length (one to sixteen bits) to be shifted into and out of the device at a programmed bit-transfer rate.
- **SPRUG03** <u>TMS320x2833x, 2823x Inter-Integrated Circuit (I2C) Reference Guide</u> describes the features and operation of the inter-integrated circuit (I2C) module.

#### Tools Guides—

- SPRU513— <u>TMS320C28x Assembly Language Tools User's Guide</u> describes the assembly language tools (assembler and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C28x device.
- SPRU514— <u>TMS320C28x Optimizing C Compiler User's Guide</u> describes the TMS320C28x<sup>™</sup> C/C++ compiler. This compiler accepts ANSI standard C/C++ source code and produces TMS320 DSP assembly language source code for the TMS320C28x device.
- SPRU608— The TMS320C28x Instruction Set Simulator Technical Overview describes the simulator, available within the Code Composer Studio for TMS320C2000 IDE, that simulates the instruction set of the C28x<sup>™</sup> core.
- **SPRU625** <u>TMS320C28x DSP/BIOS Application Programming Interface (API) Reference Guide</u> describes development using DSP/BIOS.



## Introduction

The TMS320C2000<sup>™</sup> DSP family consists of fixed-point and floating-point digital signal controllers (DSCs). TMS320C2000<sup>™</sup> Digital Signal Controllers combine control peripheral integration and ease of use of a microcontroller (MCU) with the processing power and C efficiency of TI's leading DSP technology. This chapter provides an overview of the architectural structure and components of the C28x plus floating-point unit CPU.

Торіс		Page
1.1	Introduction to the Central Processing Unit (CPU)	Q
1.2	Compatibility with the C28x Fixed-Point CPU	
1.3	Components of the C28x plus Floating-Point CPU	
1.4	Memory Interface	
	•	

7



#### 1.1 Introduction to the Central Processing Unit (CPU)

The C28x plus floating-point (C28x+FPU) processor extends the capabilities of the C28x fixed-point CPU by adding registers and instructions to support IEEE single-precision floating point operations. This device draws from the best features of digital signal processing; reduced instruction set computing (RISC); and microcontroller architectures, firmware, and tool sets. The DSC features include a modified Harvard architecture and circular addressing. The RISC features are single-cycle instruction execution, register-to-register operations, and modified Harvard architecture (usable in Von Neumann mode). The microcontroller features include ease of use through an intuitive instruction set, byte packing and unpacking, and bit manipulation. The modified Harvard architecture of the CPU enables instruction and data fetches to be performed in parallel. The CPU can read instructions and data while it writes data simultaneously to maintain the single-cycle instruction operation across the pipeline. The CPU does this over six separate address/data buses.

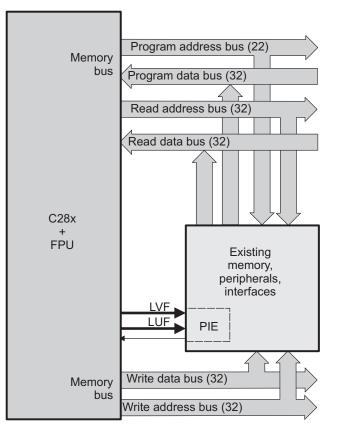
Throughout this document the following notations are used:

- C28x refers to the C28x fixed-point CPU.
- C28x plus Floating-Point and C28x+FPU both refer to the C28x CPU with enhancements to support IEEE single-precision floating-point operations.

#### 1.2 Compatibility with the C28x Fixed-Point CPU

No changes have been made to the C28x base set of instructions, pipeline, or memory bus architecture. Therefore, programs written for the C28x CPU are completely compatible with the C28x+FPU and all of the features of the C28x documented in *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number <u>SPRU430</u>) apply to the C28x+FPU.

Figure 1-1 shows basic functions of the FPU.



#### Figure 1-1. FPU Functional Block Diagram



#### 1.2.1 Floating-Point Code Development

When developing C28x floating-point code use Code Composer Studio 3.3, or later, with at least service release 8. The C28x compiler V5.0, or later, is also required to generate C28x native floating-point opcodes. This compiler is available via Code Composer Studio update advisor as a seperate download. V5.0 can generate both fixed-point as well as floating-point code. To build floating-point code use the compiler switches:-v28 and - -float\_support = fpu32. In Code Composer Studio 3.3 the float\_support option is in the build options under compiler-> advanced: floating point support. Without the float\_support flag, or with float\_support = none, the compiler will generate fixed-point code.

When building for C28x floating-point make sure all associated libraries have also been built for floating-point. The standard run-time support (RTS) libaries built for floating-point included with the compiler have fpu32 in their name. For example rts2800\_fpu32.lib and rts2800\_fpu\_eh.lib have been built for the floating-point unit. The "eh" version has exception handling for C++ code. Using the fixed-point RTS libraries in a floating-point project will result in the linker issuing an error for incompatible object files.

To improve performance of native floating-point projects, consider using the *C28x FPU Fast RTS Library* (<u>SPRC664</u>). This library contains hand-coded optimized math routines such as division, square root, atan2, sin and cos. This library can be linked into your project before the standard runtime support library to give your application a performance boost. As an example, the standard RTS library uses a polynomial expansion to calculate the sin function. The Fast RTS library, however, uses a math look-up table in the boot ROM of the device. Using this look-up table method results in approximately a 20 cycle savings over the standard RTS calculation.

#### 1.3 Components of the C28x plus Floating-Point CPU

The C28x+FPU contains:

- A central processing unit for generating data and program-memory addresses; decoding and executing instructions; performing arithmetic, logical, and shift operations; and controlling data transfers among CPU registers, data memory, and program memory
- A floating-point unit for IEEE single-precision floating point operations.
- Emulation logic for monitoring and controlling various parts and functions of the device and for testing device operation. This logic is identical to that on the C28x fixed-point CPU.
- Signals for interfacing with memory and peripherals, clocking and controlling the CPU and the emulation logic, showing the status of the CPU and the emulation logic, and using interrupts. This logic is identical to the C28x fixed-point CPU.

Some features of the C28x+FPU central processing unit are:

- Fixed-Point instructions are pipeline protected. This pipeline for fixed-point instructions is identical to that on the C28x fixed-point CPU. The CPU implements an 8-phase pipeline that prevents a write to and a read from the same location from occurring out of order. See Figure 3-1
- Some floating-point instructions require pipeline alignment. This alignment is done through software to allow the user to improve performance by taking advantage of required delay slots.
- Independent register space. These registers function as system-control registers, math registers, and data pointers. The system-control registers are accessed by special instructions.
- Arithmetic logic unit (ALU). The 32-bit ALU performs 2s-complement arithmetic and Boolean logic operations.
- Floating point unit (FPU). The 32-bit FPU performs IEEE single-precision floating-point operations.
- Address register arithmetic unit (ARAU). The ARAU generates data memory addresses and increments or decrements pointers in parallel with ALU operations.
- Barrel shifter. This shifter performs all left and right shifts of fixed-point data. It can shift data to the left by up to 16 bits and to the right by up to 16 bits.
- Fixed-Point Multiplier. The multiplier performs 32-bit × 32-bit 2s-complement multiplication with a 64-bit result. The multiplication can be performed with two signed numbers, two unsigned numbers, or one signed number and one unsigned number.



#### 1.3.1 Emulation Logic

The emulation logic is identical to that on the C28x fixed-point CPU. This logic includes the following features. For more details about these features, refer to the *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430:

- Debug-and-test direct memory access (DT-DMA). A debug host can gain direct access to the content
  of registers and memory by taking control of the memory interface during unused cycles of the
  instruction pipeline.
- A counter for performance benchmarking.
- Multiple debug events. Any of the following debug events can cause a break in program execution:
  - A breakpoint initiated by the ESTOP0 or ESTOP1 instruction.
  - An access to a specified program-space or data-space location.
     When a debug event causes the C28x to enter the debug-halt state, the event is called a break event.
- Real-time mode of operation.

#### 1.3.2 Memory Map

Like the C28x, the C28x+FPU uses 32-bit data addresses and 22-bit program addresses. This allows for a total address reach of 4G words (1 word = 16 bits) in data space and 4M words in program space. Memory blocks on all C28x+FPU designs are uniformly mapped to both program and data space. For specific details about each of the map segments, see the data sheet for your device.

#### 1.3.3 On-Chip Program and Data

All C28x+FPU based devices contain at least two blocks of single access on-chip memory referred to as M0 and M1. Each of these blocks is 1K words in size. M0 is mapped at addresses 0x0000 – 0x03FF and M1 is mapped at addresses 0x0400 – 0x07FF. Like all other memory blocks on the C28x+FPU devices, M0 and M1 are mapped to both program and data space. Therefore, you can use M0 and M1 to execute code or for data variables. At reset, the stack pointer is set to the top of block M1. Depending on the device, it may also have additional random-access memory (RAM), read-only memory (ROM), external interface zones, or flash memory.

#### 1.3.4 CPU Interrupt Vectors

The C28x+FPU interrupt vectors are identical to those on the C28x CPU. Sixty-four addresses in program space are set aside for a table of 32 CPU interrupt vectors. The CPU vectors can be mapped to the top or bottom of program space by way of the VMAP bit. For more information about the CPU vectors, see *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number <u>SPRU430</u>). For devices with a peripheral interrupt expansion (PIE) block, the interrupt vectors will reside in the PIE vector table and this memory can be used as program memory.

#### 1.4 Memory Interface

The C28x+FPU memory interface is identical to that on the C28x. The C28x+FPU memory map is accessible outside the CPU by the memory interface, which connects the CPU logic to memories, peripherals, or other interfaces. The memory interface includes separate buses for program space and data space. This means an instruction can be fetched from program memory while data memory is being accessed. The interface also includes signals that indicate the type of read or write being requested by the CPU. These signals can select a specified memory block or peripheral for a given bus transaction. In addition to 16-bit and 32-bit accesses, the C28x+FPU supports special byte-access instructions that can access the least significant byte (LSByte) or most significant byte (MSByte) of an addressed word. Strobe signals indicate when such an access is occurring on a data bus.



#### 1.4.1 Address and Data Buses

Like the C28x, the memory interface has three address buses:

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

The memory interface also has three data buses:

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

The 32-bit DWDB carries data during writes to data space or program space.

A program-space read and a program-space write cannot happen simultaneously because both use the PAB. Similarly, a program-space write and a data-space write cannot happen simultaneously because both use the DWDB. Transactions that use different buses can happen simultaneously. For example, the CPU can read from program space (using PAB and PRDB), read from data space (using DRAB and DRDB), and write to data space (using DWAB and DWDB) at the same time. This behavior is identical to the C28x CPU.

#### 1.4.2 Alignment of 32-Bit Accesses to Even Addresses

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

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

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



Chapter 2 SPRUE02A–June 2007–Revised August 2008

## **CPU Register Set**

The C28x+FPU architecture is the same as the C28x CPU with an extended register and instruction set to support IEEE single-precision floating point operations. This section describes the extensions to the C28x architecture.

Торіс		Page
2.1	CPU Registers	14



#### 2.1 CPU Registers

Devices with the C28x+FPU include the standard C28x register set plus an additional set of floating-point unit registers. The additional floating-point unit registers are the following:

- Eight floating-point result registers, RnH (where n = 0 7)
- Floating-point Status Register (STF)
- Repeat Block Register (RB)

All of the floating-point registers except the repeat block register are shadowed. This shadowing can be used in high priority interrupts for fast context save and restore of the floating-point registers.

Figure 2-1 shows a diagram of both register sets and Table 2-1 shows a register summary. For information on the standard C28x register set, see the *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number <u>SPRU430</u>).

Standard C28x Register Set	Additional 32-bit FPU Registers
ACC (32-bit)	R0H (32-bit)
P (32-bit)	
XT (32-bit)	R1H (32-bit)
XAR0 (32-bit)	R2H (32-bit)
XAR1 (32-bit)	R3H (32-bit)
XAR2 (32-bit)	
XAR3 (32-bit)	R4H (32-bit)
XAR4 (32-bit)	R5H (32-bit)
XAR5 (32-bit)	
XAR6 (32-bit)	R6H (32-bit)
XAR7 (32-bit)	R7H (32-bit)
PC (22-bit)	
RPC (22-bit)	FPU Status Register (STF)
DP (16-bit)	Repeat Block Register (RB)
SP (16-bit)	FPU registers R0H - R7H and STF
ST0 (16 hit)	are shadowed for fast context save and restore
ST0 (16-bit)	
ST1 (16-bit)	
IER (16-bit)	
IFR (16-bit)	
DBGIER (16-bit)	

#### Figure 2-1. C28x With Floating-Point Registers



Table 2-1. 28x Plus Floating-Point CPU Register Summar	v
Tuble E 1. Eox 1 lus 1 louding 1 onit of o Register Ourinnar	y

Register	C28x CPU	C28x+FPU	Size	Description	Value After Reset
ACC	Yes	Yes	32 bits	Accumulator	0x0000000
AH	Yes	Yes	16 bits	High half of ACC	0x0000
AL	Yes	Yes	16 bits	Low half of ACC	0x0000
XAR0	Yes	Yes	16 bits	Auxiliary register 0	0x0000000
XAR1	Yes	Yes	32 bits	Auxiliary register 1	0x0000000
XAR2	Yes	Yes	32 bits	Auxiliary register 2	0x0000000
XAR3	Yes	Yes	32 bits	Auxiliary register 3	0x0000000
XAR4	Yes	Yes	32 bits	Auxiliary register 4	0x0000000
XAR5	Yes	Yes	32 bits	Auxiliary register 5	0x0000000
XAR6	Yes	Yes	32 bits	Auxiliary register 6	0x0000000
XAR7	Yes	Yes	32 bits	Auxiliary register 7	0x0000000
AR0	Yes	Yes	16 bits	Low half of XAR0	0x0000
AR1	Yes	Yes	16 bits	Low half of XAR1	0x0000
AR2	Yes	Yes	16 bits	Low half of XAR2	0x0000
AR3	Yes	Yes	16 bits	Low half of XAR3	0x0000
AR4	Yes	Yes	16 bits	Low half of XAR4	0x0000
AR5	Yes	Yes	16 bits	Low half of XAR5	0x0000
AR6	Yes	Yes	16 bits	Low half of XAR6	0x0000
AR7	Yes	Yes	16 bits	Low half of XAR7	0x0000
DP	Yes	Yes	16 bits	Data-page pointer	0x0000
IFR	Yes	Yes	16 bits	Interrupt flag register	0x0000
IER	Yes	Yes	16 bits	Interrupt enable register	0x0000
DBGIER	Yes	Yes	16 bits	Debug interrupt enable register	0x0000
P	Yes	Yes	32 bits	Product register	0x0000000
PH	Yes	Yes	16 bits	High half of P	0x0000
PL	Yes	Yes	16 bits	Low half of P	0x0000
PC	Yes	Yes	22 bits	Program counter	0x3FFFC0
RPC	Yes	Yes	22 bits	Return program counter	0x0000000
SP	Yes	Yes	16 bits	Stack pointer	0x0400
ST0	Yes	Yes	16 bits	Status register 0	0x0000
ST1	Yes	Yes	16 bits	Status register 1	0x080B <sup>(1)</sup>
хт	Yes	Yes	32 bits	Multiplicand register	0x0000000
т	Yes	Yes	16 bits	High half of XT	0x0000
TL	Yes	Yes	16 bits	Low half of XT	0x0000
ROH	No	Yes	32 bits	Floating-point result register 0	0.0
R1H	No	Yes	32 bits	Floating-point result register 1	0.0
R2H	No	Yes	32 bits	Floating-point result register 2	0.0
R3H	No	Yes	32 bits	Floating-point result register 3	0.0
R4H	No	Yes	32 bits	Floating-point result register 4	0.0
R5H	No	Yes	32 bits	Floating-point result register 5	0.0
R6H	No	Yes	32 bits	Floating-point result register 6	0.0
R7H	No	Yes	32 bits	Floating-point result register 7	0.0
STF	No	Yes	32 bits	Floating-point status register	0x0000000
RB	No	Yes	32 bits	Repeat block register	0x0000000

<sup>(1)</sup> Reset value shown is for devices without the VMAP signal and MOM1MAP signal pinned out. On these devices both of these signals are tied high internal to the device.



#### 2.1.1 Floating-Point Status Register (STF)

The floating-point status register (STF) reflects the results of floating-point operations. There are three basic rules for floating point operation flags:

- 1. Zero and negative flags are set based on moves to registers.
- 2. Zero and negative flags are set based on the result of compare, minimum, maximum, negative and absolute value operations.
- 3. Overflow and underflow flags are set by math instructions such as multiply, add, subtract and 1/x. These flags may also be connected to the peripheral interrupt expansion (PIE) block on your device. This can be useful for debugging underflow and overflow conditions within an application.

As on the C28x, program flow is controlled by C28x instructions that read status flags in the status register 0 (ST0). If a decision needs to be made based on a floating-point operation, the information in the STF register needs to be loaded into ST0 flags (Z,N,OV,TC,C) so that the appropriate branch conditional instruction can be executed. The MOVST0 FLAG instruction is used to load the current value of specified STF flags into the respective bits of ST0. When this instruction executes, it will also clear the latched overflow and underflow flags if those flags are specified.

#### Example 2-1. Moving STF Flags to the ST0 Register

```
Loop:

MOV32 R0H,*XAR4++

MOV32 R1H,*XAR3++

CMPF32 R1H, R0H

MOVST0 ZF, NF ; Move ZF and NF to ST0

BF Loop, GT ; Loop if (R1H > R0H)
```

#### 31 30 16 SHDWS Reserved R/W-0 R-0 15 10 9 8 7 6 5 4 3 2 0 1 Reserved RND32 Reserved TF ΖI NI 7F NF LUF LVF R-0 R/W-0 R-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0

#### Figure 2-2. Floating-point Unit Status Register (STF)

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

#### Table 2-2. Floating-point Unit Status (STF) Register Field Descriptions

Bits	Field	Value	Description
31	SHDWS		Shadow Mode Status Bit
		0	This bit is forced to 0 by the RESTORE instruction.
		1	This bit is set to 1 by the SAVE instruction.
			This bit is not affected by loading the status register either from memory or from the shadow values.
30 - 10	Reserved	0	Reserved for future use
9	RND32		Round 32-bit Floating-Point Mode
		0	If this bit is zero, the MPYF32, ADDF32 and SUBF32 instructions will round to zero (truncate).
		1	If this bit is one, the MPYF32, ADDF32 and SUBF32 instructions will round to the nearest even value.
8 - 7	Reserved	0	Reserved for future use



#### Table 2-2. Floating-point Unit Status (STF) Register Field Descriptions (continued)

Bits	Field	Value	Description
6	TF		Test Flag
			The TESTTF instruction can modify this flag based on the condition tested. The SETFLG and SAVE instructions can also be used to modify this flag.
		0	The condition tested with the TESTTF instruction is false.
		1	The condition tested with the TESTTF instruction is true.
5	ZI		Zero Integer Flag
			The following instructions modify this flag based on the integer value stored in the destination register: MOV32, MOVD32, MOVDD32 The SETFLG and SAVE instructions can also be used to modify this flag.
		0	The integer value is not zero.
		1	The integer value is zero.
4	NI		Negative Integer Flag
			The following instructions modify this flag based on the integer value stored in the destination register: MOV32, MOVD32, MOVDD32 The SETFLG and SAVE instructions can also be used to modify this flag.
		0	The integer value is not negative.
		1	The integer value is negative.
3	ZF		Zero Floating-Point Flag <sup>(1)(2)</sup>
-			The following instructions modify this flag based on the floating-point value stored in the destination
			register: MOV32, MOVD32, MOVDD32, ABSF32, NEGF32 The CMPF32, MAXF32, and MINF32 instructions modify this flag based on the result of the operation. The SETFLG and SAVE instructions can also be used to modify this flag
		0	The floating-point value is not zero.
		1	The floating-point value is zero.
2	NF		Negative Floating-Point Flag <sup>(1)(2)</sup>
			The following instructions modify this flag based on the floating-point value stored in the destination register: MOV32, MOVD32, MOVDD32, ABSF32, NEGF32 The CMPF32, MAXF32, and MINF32 instructions modify this flag based on the result of the operation. The SETFLG and SAVE instructions can also be used to modify this flag.
		0	The floating-point value is not negative.
		1	The floating-point value is negative.
1	LUF		Latched Underflow Floating-Point Flag
			The following instructions will set this flag to 1 if an underflow occurs: MPYF32, ADDF32, SUBF32, MACF32, EINVF32, EISQRTF32
		0	An underflow condition has not been latched. If the MOVST0 instruction is used to copy this bit to ST0 then LUF will be cleared.
		1	An underflow condition has been latched.
0	LVF		Latched Overflow Floating-Point Flag
			The following instructions will set this flag to 1 if an overflow occurs: MPYF32, ADDF32, SUBF32, MACF32, EINVF32, EISQRTF32
		0	An overflow condition has not been latched. If the MOVST0 instruction is used to copy this bit to ST0, then LVF will be cleared.
		1	An overflow condition has been latched.

A negative zero floating-point value is treated as a positive zero value when configuring the ZF and NF flags. A DeNorm floating-point value is treated as a positive zero value when configuring the ZF and NF flags. (1)

(2)



CPU Registers

#### 2.1.2 Repeat Block Register (RB)

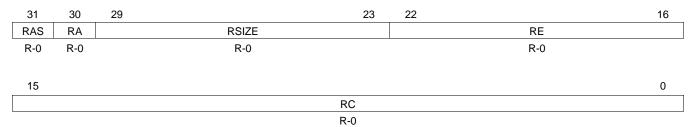
The repeat block instruction (RPTB) is a new instruction for C28x+FPU. This instruction allows you to repeat a block of code as shown in Example 2-2.

#### Example 2-2. The Repeat Block (RPTB) Instruction uses the RB Register

; find the largest element and p MOV32 R0H, *XAR0++;	out its address in XAR6
.align 2	; Aligns the next instruction to an even address
NOP	; Makes RPTB odd aligned - required for a block size of 8
RPTB VECTOR_MAX_END, AR7	; RA is set to 1
MOVL ACC, XAR0	
MOV32 R1H,*XAR0++	; RSIZE reflects the size of the RPTB block
MAXF32 R0H,R1H	; in this case the block size is 8
MOVSTO NF, ZF	
MOVL XAR6, ACC, LT	
VECTOR_MAX_END:	; RE indicates the end address. RA is cleared

The C28x\_FPU hardware automatically populates the RB register based on the execution of a RPTB instruction. This register is not normally read by the application and does not accept debugger writes.

#### Figure 2-3. Repeat Block Register (RB)



LEGEND: R = Read only; -n = value after reset

#### Table 2-3. Repeat Block (RB) Register Field Descriptions

Bits	Field	Value	Description
31	RAS		Repeat Block Active Shadow Bit
			When an interrupt occurs the repeat active, RA, bit is copied to the RAS bit and the RA bit is cleared. When an interrupt return instruction occurs, the RAS bit is copied to the RA bit and RAS is cleared.
		0	A repeat block was not active when the interrupt was taken.
		1	A repeat block was active when the interrupt was taken.
30	RA		Repeat Block Active Bit
		0	This bit is cleared when the repeat counter, RC, reaches zero.
			When an interrupt occurs the RA bit is copied to the repeat active shadow, RAS, bit and RA is cleared. When an interrupt return, IRET, instruction is executed, the RAS bit is copied to the RA bit and RAS is cleared.
		1	This bit is set when the RPTB instruction is executed to indicate that a RPTB is currently active.
29-23	RSIZE		Repeat Block Size
			This 7-bit value specifies the number of 16-bit words within the repeat block. This field is initialized when the RPTB instruction is executed. The value is calculated by the assembler and inserted into the RPTB instruction's RSIZE opcode field.
		0-7	Illegal block size.
		8/9-0x7F	A RPTB block that starts at an even address must include at least 9 16-bit words and a block that starts at an odd address must include at least 8 16-bit words. The maximum block size is 127 16-bit words. The codegen assembler will check for proper block size and alignment.



Table 2-3	. Repeat Block	(RB) Register Fi	ield Descriptions	(continued)
		(		(

Bits	Field	Value	Description
22-16	RE		Repeat Block End Address
			This 7-bit value specifies the end address location of the repeat block. The RE value is calculated by hardware based on the RSIZE field and the PC value when the RPTB instruction is executed.
			RE = lower 7 bits of (PC + 1 + RSIZE)
15-0	RC		Repeat Count
		0	The block will not be repeated; it will be executed only once. In this case the repeat active, RA, bit will not be set.
		1- 0xFFFF	This 16-bit value determines how many times the block will repeat. The counter is initialized when the RPTB instruction is executed and is decremented when the PC reaches the end of the block. When the counter reaches zero, the repeat active bit is cleared and the block will be executed one more time. Therefore the total number of times the block is executed is RC+1.



## Pipeline

The pipeline flow for C28x instructions is identical to that of the C28x CPU described in *TMS320C28x DSP CPU and Instruction Set Reference Guide* (SPRU430). Some floating-point instructions, however, use additional execution phases and thus require a delay to allow the operation to complete. This pipeline alignment is achieved by inserting NOPs or non-conflicting instructions when required. Software control of delay slots allows you to improve performance of an application by taking advantage of the delay slots and filling them with non-conflicting instructions. This section describes the key characteristics of the pipeline with regards to floating-point instructions. The rules for avoiding pipeline conflicts are small in number and simple to follow and the C28x+FPU assembler will help you by issuing errors for conflicts.

Горіс				
3.1	Pipeline Overview	22		
3.2	General Guidelines for Floating-Point Pipeline Alignment	22		
3.3	Moves from FPU Registers to C28x Registers	23		
3.4	Moves from C28x Registers to FPU Registers	23		
3.5	Parallel Instructions	24		
3.6	Invalid Delay Instructions	24		
3.7	Optimizing the Pipeline	27		



#### 3.1 Pipeline Overview

The C28x FPU pipeline is identical to the C28x pipeline for all standard C28x instructions. In the decode2 stage (D2), it is determined if an instruction is a C28x instruction or a floating-point unit instruction. The pipeline flow is shown in Figure 3-1. Notice that stalls due to normal C28x pipeline stalls (D2) and memory waitstates (R2 and W) will also stall any C28x FPU instruction. Most C28x FPU instructions are single cycle and will complete in the FPU E1 or W stage which aligns to the C28x pipeline. Some instructions will take an additional execute cycle (E2). For these instructions you must wait a cycle for the result from the instruction to be available. The rest of this section will describe when delay cycles are required. Keep in mind that the assembly tools for the C28x+FPU will issue an error if a delay slot has not been handled correctly.

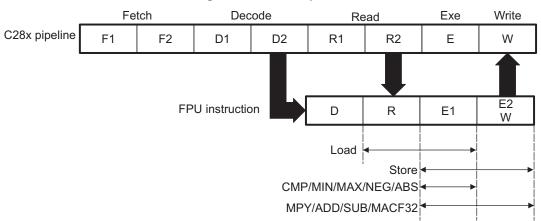


Figure 3-1. FPU Pipeline

#### 3.2 General Guidelines for Floating-Point Pipeline Alignment

While the C28x+FPU assembler will issue errors for pipeline conflicts, you may still find it useful to understand when software delays are required. This section describes three guidelines you can follow when writing C28x+FPU assembly code.

Floating-point instructions that require delay slots have a 'p' after their cycle count. For example '2p' stands for 2 pipelined cycles. This means that an instruction can be started every cycle, but the result of the instruction will only be valid one instruction later.

There are three general guidelines to determine if an instruction needs a delay slot:

- 1. Floating-point math operations (multiply, addition, subtraction, 1/x and MAC) require 1 delay slot.
- 2. Conversion instructions between integer and floating-point formats require 1 delay slot.
- 3. Everything else does not require a delay slot. This includes minimum, maximum, compare, load, store, negative and absolute value instructions.

There are two exceptions to these rules. First, moves between the CPU and FPU registers require special pipeline alignment that is described later in this section. These operations are typically infrequent. Second, the MACF32 R7H, R3H, mem32, \*XAR7 instruction has special requirements that make it easier to use. Refer to the MACF32 instruction description for details.

An example of the 32-bit ADDF32 instruction is shown in Example 3-1. ADDF32 is a 2p instruction and therefore requires one delay slot. The destination register for the operation, R0H, will be updated one cycle after the instruction for a total of 2 cycles. Therefore, a NOP or instruction that does not use R0H must follow this instruction.

Any memory stall or pipeline stall will also stall the floating-point unit. This keeps the floating-point unit aligned with the C28x pipeline and there is no need to change the code based on the waitstates of a memory block.



#### Example 3-1. 2p Instruction Pipeline Alignment

ADDF32 R0H, #1.5, R1H	; 2 pipeline cycles (2p)
NOP	; 1 cycle delay or non-conflicting instruction
	; < ADDF32 completes, ROH updated
NOP	; Any instruction

#### 3.3 Moves from FPU Registers to C28x Registers

When transferring from the floating-point unit registers to the C28x CPU registers, additional pipeline alignment is required as shown in Example 3-2 and Example 3-3.

Example 3-2. Floating-Point to C28x Register Software Pipeline Alignment

; ;			51	point minimum: single-cycle operation quired before copying ROH to ACC
	MINF32	ROH,	R1H	; Single-cycle instruction ; < ROH is valid
	NOP MOV32	@ACC	, R0H	; Alignment cycle ; Copy ROH to ACC

For 1-cycle FPU instructions, one delay slot is required between a write to the floating-point register and the transfer instruction as shown in Example 3-2. For 2p FPU instructions, two delay slots are required between a write to the floating-point register and the transfer instruction as shown in Example 3-3.

#### Example 3-3. Floating-Point to C28x Register Software Pipeline Alignment

;;	ADDF32: 32-bit floating-point addition: 2p operation An alignment cycle is required before copying ROH to ACC				
	ADDF32	ROH,	л R1H, #2	- ;	ROH = R1H + 2, 2 pipeline cycle instruction
	NOP			;	1 delay cycle or non-conflicting instruction
				;	< ROH is valid
	NOP			;	Alignment cycle
	MOV32	@ACC,	ROH	;	Copy ROH to ACC

#### 3.4 Moves from C28x Registers to FPU Registers

Transfers from the standard C28x CPU registers to the floating-point registers require four alignment cycles. In this case the four alignment cycles can be filled with NOPs or any non-conflicting instruction except for FRACF32, UI16TOF32, I16TOF32, F32TOUI32, and F32TOI32. These instructions cannot replace any of the four alignment NOPs.

#### Example 3-4. C28x Register to Floating-Point Register Software Pipeline Alignment

```
; Four alignment cycles are required after copying a standard 28x CPU
; register to a floating-point register.
;
MOV32 R0H,@ACC ; Copy ACC to R0H
NOP
NOP
NOP
NOP
NOP
NOP
; Wait 4 cycles
ADDF32 R2H,R1H,R0H ; R0H is valid
```



#### 3.5 Parallel Instructions

Parallel instructions are single opcodes that perform two operations in parallel. This can be a math operation in parallel with a move operation, or two math operations in parallel. Math operations with a parallel move are referred to as 2p/1 instructions. The math portion of the operation takes 2 pipelined cycles while the move portion of the operation is single cycle. This means that NOPs or other non conflicting instructions must be inserted to align the math portion of the operation. An example of an add with parallel move instruction is shown in Example 3-5.

#### Example 3-5. 2p/1 Parallel Instruction Software Pipeline Alignment

```
ADDF32 || MOV32 instruction: 32-bit floating-point add with parallel move
;
  ADDF32 is a 2p operation
;
;
  MOV32 is a 1 cycle operation
;
  ADDF32 R0H, R1H, #2
                             ; ROH = R1H + 2, 2 pipeline cycle operation
|| MOV32 R1H, @Val
                            ; R1H gets the contents of Val, single cycle operation
                            ; <-- MOV32 completes here (R1H is valid)
  NOP
                             ; 1 cycle delay or non-conflicting instruction
                             ; <-- ADDF32 completes here (ROH is valid)
  NOP
                             ; Any instruction
```

Parallel math instructions are referred to as 2p/2p instructions. Both math operations take 2 cycles to complete. This means that NOPs or other non conflicting instructions must be inserted to align the both math operations. An example of a multiply with parallel add instruction is shown in Example 3-5

#### Example 3-6. 2p/2p Parallel Instruction Software Pipeline Alignment

```
; MPYF32 || ADDF32 instruction: 32-bit floating-point multiply with parallel add
; MPYF32 is a 2p operation
; ADDF32 is a 2p cycle operation
;
MPYF32 ROH, R1H, R3H ; R0H = R1H * R3H, 2 pipeline cycle operation
|| ADDF32 R1H, R2H, R4H ; R1H = R2H + R4H, 2 pipeline cycle operation
NOP ; 1 cycle delay or non-conflicting instruction
; <--- MPYF32 and ADDF32 complete here (R0H and R1H are valid)
NOP ; Any instruction
```

#### 3.6 Invalid Delay Instructions

Most instructions can be used in delay slots as long as source and destination register conflicts are avoided. The C28x+FPU assembler will issue an error anytime you use an conflicting instruction within a delay slot. The following guidelines can be used to avoid these conflicts.

**Note:** Destination register conflicts in delay slots:

Any operation used for pipeline alignment delay must not use the same destination register as the instruction requiring the delay. See Example 3-7.

In Example 3-7 the MPYF32 instruction uses R2H as its destination register. The next instruction should not use R2H as its destination. Since the MOV32 instruction uses the R2H register a pipeline conflict will be issued by the assembler. This conflict can be resolved by using a register other than R2H for the MOV32 instruction as shown in Example 3-8.



#### Example 3-7. Destination Register Conflict

```
; Invalid delay instruction. Both instructions use the same destination register
MPYF32 R2H, R1H, R0H ; 2p instruction
MOV32 R2H, mem32 ; Invalid delay instruction
```

#### Example 3-8. Destination Register Conflict Resolved

```
; Valid delay instruction
MPYF32 R2H, R1H, R0H ; 2p instruction
MOV32 R1H, mem32 ; Valid delay
; <-- MPYF32 completes, R2H valid</pre>
```

**Note:** Instructions in delay slots cannot use the instruction's destination register as a source register.

Any operation used for pipeline alignment delay must not use the destination register of the instruction requiring the delay as a source register as shown in Example 3-9. For parallel instructions, the current value of a register can be used in the parallel operation before it is overwritten as shown in Example 3-11.

In Example 3-9 the MPYF32 instruction again uses R2H as its destination register. The next instruction should not use R2H as its source since the MPYF32 will take an additional cycle to complete. Since the ADDF32 instruction uses the R2H register a pipeline conflict will be issued by the assembler. This conflict can be resolved by using a register other than R2H or by inserting a non-conflicting instruction between the MPYF32 and ADDF32 instructions. Since the SUBF32 does not use R2H this instruction can be moved before the ADDF32 as shown in Example 3-10.

#### Example 3-9. Destination/Source Register Conflict

```
; Invalid delay instruction. ADDF32 should not use R2H as a source operand
MPYF32 R2H, R1H, R0H ; 2p instruction
ADDF32 R3H, R3H, R2H ; Invalid delay instruction
SUBF32 R4H, R1H, R0H
```

#### Example 3-10. Destination/Source Register Conflict Resolved

```
; Valid delay instruction.
MPYF32 R2H, R1H, R0H
SUBF32 R4H, R1H, R0H
ADDF32 R3H, R3H, R2H
NOP
; <-- SUBF32 completes, R4H valid</pre>
```

It should be noted that a source register for the 2nd operation within a parallel instruction can be the same as the destination register of the first operation. This is because the two operations are started at the same time. The 2nd operation is not in the delay slot of the first operation. Consider Example 3-11 where the MPYF32 uses R2H as its destination register. The MOV32 is the 2nd operation in the instruction and can freely use R2H as a source register. The contents of R2H before the multiply will be used by MOV32.



#### Example 3-11. Parallel Instruction Destination/Source Exception

; Valid parallel operation.	
MPYF32 R2H, R1H, R0H	; 2p/1 instruction
MOV32 mem32, R2H	; < Uses R2H before the MPYF32
	; < mem32 updated
NOP	; < Delay for MPYF32
	; < R2H updated
	-

Likewise, the source register for the 2nd operation within a parallel instruction can be the same as one of the source registers of the first operation. The MPYF32 operation in Example 3-12 uses the R1H register as one of its sources. This register is also updated by the MOV32 register. The multiplication operation will use the value in R1H before the MOV32 updates it.

#### Example 3-12. Parallel Instruction Destination/Source Exception

```
; Valid parallel instruction
MPYF32 R2H, R1H, R0H ; 2p/1 instruction
|| MOV32 R1H, mem32 ; Valid
NOP ; <-- MOV32 completes, R1H valid
; <-- MPYF32, R2H valid</pre>
```

**Note:** Operations within parallel instructions cannot use the same destination register.

When two parallel operations have the same destination register, the result is invalid.

For example, see Example 3-13.

If both operations within a parallel instruction try to update the same destination register as shown in Example 3-13 the assembler will issue an error.

#### Example 3-13. Invalid Destination Within a Parallel Instruction

```
; Invalid parallel instruction. Both operations use the same destination register
MPYF32 R2H, R1H, R0H ; 2p/1 instruction
|| MOV32 R2H, mem32 ; Invalid
```

Some instructions access or modify the STF flags. Because the instruction requiring a delay slot will also be accessing the STF flags, these instructions should not be used in delay slots. These instructions are SAVE, SETFLG, RESTORE and MOVST0.

**Note:** Do not use SAVE, SETFLG, RESTORE, or the MOVST0 instruction in a delay slot.



#### 3.7 Optimizing the Pipeline

The following example shows how delay slots can be used to improve the performance of an algorithm. The example performs two Y = MX+B operations. In Example 3-14, no optimization has been done. The Y = MX+B calculations are sequential and each takes 7 cycles to complete. Notice there are NOPs in the delay slots that could be filled with non-conflicting instructions. The only requirement is these instructions must not cause a register conflict or access the STF register flags.

Example 3-14. Floating-Point Code Without Pipeline Optimization

```
; Using NOPs for alignment cycles, calculate the following:
; Y1 = M1 * X1 + B1
 Y2 = M2 * X2 + B2
;
;
; Calculate Y1
  NOV32RUH,@M1; Load ROH with M1 - single cycleMOV32R1H,@X1; Load R1H with X1 - single cycleMPYF32R1H,R1H,ROH; R1H = M1 * X1 - 20 cmMOV32R0H,@B1;
;
|| MOV32 R0H,@B1
   NOP
                               ; Wait for MPYF32 to complete
                                ; <-- MPYF32 completes, R1H is valid
   ADDF32 R1H,R1H,R0H
                                ; R1H = R1H + R0H - 2p operation
   NOP
                               ; Wait for ADDF32 to complete
                                ; <-- ADDF32 completes, R1H is valid
   MOV32 @Y1,R1H
                                 ; Save R1H in Y1
                                                     - single cycle
; Calculate Y2
   MOV32
           ROH,@M2
                                ; Load ROH with M2 - single cycle
   MOV32
           R1H,@X2
                               ; Load R1H with X2 - single cycle
                                ; R1H = M2 * X2 - 2p operation
; Load R0H with B2 - single cycle
   MPYF32 R1H,R1H,R0H
|| MOV32
           R0H,@B2
   NOP
                                ; Wait for MPYF32 to complete
                                ; <-- MPYF32 completes, R1H is valid
   ADDF32 R1H,R1H,R0H
                                ; R1H = R1H + R0H
                                ; Wait for ADDF32 to complete
   NOP
                                ; <-- ADDF32 completes, R1H is valid
   MOV32 @Y2,R1H
                                 ; Save R1H in Y2
; 14 cycles
; 48 bytes
```

The code shown in Example 3-15 was generated by the C28x+FPU compiler with optimization enabled. Notice that the NOPs in the first example have now been filled with other instructions. The code for the two Y = MX+B calculations are now interleaved and both calculations complete in only 9 cycles.



#### Example 3-15. Floating-Point Code With Pipeline Optimization

```
; Using non-conflicting instructions for alignment cycles,
; calculate the following:
; Y1 = M1*X1 + B1
; Y2 = M2 * X2 + B2
   MOV32
             R2H,@X1
                              ; Load R2H with X1 - single cycle
                             ; Load R1H with M1 - single cycle
   MOV32
             R1H,@M1
                             ; R3H = M1 * X1 - 2p operation
; Load R0H with M2 - single cycle
   MPYF32
             R3H,R2H,R1H
             ROH,@M2
MOV32
                               ; Load R1H with X2 - single cycle
   MOV32
             R1H,@X2
                               ; <-- MPYF32 completes, R3H is valid
                               ; ROH = M2 * X2
   MPYF32
             ROH,R1H,ROH
                                                  - 2p operation
|| MOV32
             R4H,@B1
                              ; Load R4H with B1 - single cycle
                               ; <-- MOV32 completes, R4H is valid
                               ; R1H = B1 + M1*X1 - 2p operation
; Load R2H with B2 - single cycle
   ADDF32
             R1H,R4H,R3H
|| MOV32
             R2H,@B2
                               ; <-- MPYF32 completes, ROH is valid
                               ; R0H = B2 + M2\timesZ - 2p operation
   ADDF32
             ROH,R2H,ROH
                               ; <-- ADDF32 completes, R1H is valid
   MOV32
             @Y1,R1H
                               ; Store Y1
                               ; <-- ADDF32 completes, ROH is valid
   MOV32
             @Y2,R0H
                               ; Store Y2
;
 9 cycles
; 36 bytes
```

28



### **Instruction Set**

This chapter describes the assembly language instructions of the TMS320C28x plus floating-point processor. Also described are parallel operations, conditional operations, resource constraints, and addressing modes. The instructions listed here are an extension to the standard C28x instruction set. For information on standard C28x instructions, see the *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430).

Торіс		Page
4.1 4.2	Instruction Descriptions	



#### 4.1 Instruction Descriptions

This section gives detailed information on the instruction set. Each instruction may present the following information:

- Operands
- Opcode
- Description
- Exceptions
- Pipeline
- Examples
- See also

The example INSTRUCTION is shown to familiarize you with the way each instruction is described. The example describes the kind of information you will find in each part of the individual instruction description and where to obtain more information. On the C28x+FPU instructions, follow the same format as the C28x. The source operand(s) are always on the right and the destination operand(s) are on the left.

The explanations for the syntax of the operands used in the instruction descriptions for the TMS320C28x plus floating-point processor are given in Table 4-1. For information on the operands of standard C28x instructions, see the TMS320C28x DSP CPU and Instruction Set Reference Guide (SPRU430).

Symbol	Description
#16FHi	16-bit immediate (hex or float) value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FHiHex	16-bit immediate hex value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FLoHex	A 16-bit immediate hex value that represents the lower 16-bits of an IEEE 32-bit floating-point value
#32Fhex	32-bit immediate value that represents an IEEE 32-bit floating-point value
#32F	Immediate float value represented in floating-point representation
#0.0	Immediate zero
#RC	16-bit immediate value for the repeat count
*(0:16bitAddr)	16-bit immediate address, zero extended
CNDF	Condition to test the flags in the STF register
FLAG	Selected flags from STF register (OR) 11 bit mask indicating which floating-point status flags to change
label	Label representing the end of the repeat block
mem16	Pointer (using any of the direct or indirect addressing modes) to a 16-bit memory location
mem32	Pointer (using any of the direct or indirect addressing modes) to a 32-bit memory location
RaH	R0H to R7H registers
RbH	R0H to R7H registers
RcH	R0H to R7H registers
RdH	R0H to R7H registers
ReH	R0H to R7H registers
RfH	R0H to R7H registers
RB	Repeat Block Register
STF	FPU Status Register
VALUE	Flag value of 0 or 1 for selected flag (OR) 11 bit mask indicating the flag value; 0 or 1

#### **Table 4-1. Operand Nomenclature**



#### INSTRUCTION dest1, source1, source2 Short Description

#### Operands

•		
	dest1	description for the 1st operand for the instruction
	source1	description for the 2nd operand for the instruction
	source2	description for the 3rd operand for the instruction
		tion has a table that gives a list of the operands and a short description. always have their destination operand(s) first followed by the source
Opcode	This section	shows the opcode for the instruction.
Description		cription of the instruction execution is described. Any constraints on the processor or the assembler are discussed.
Restrictions	Any constrai discussed.	ints on the operands or use of the instruction imposed by the processor are
Pipeline	This section Chapter 3.	describes the instruction in terms of pipeline cycles as described in
Example	before and a	f instruction execution. If applicable, register and memory values are given after instruction execution. All examples assume the device is running with DE set to 1. Normally the boot ROM or the c-code initialization will set this
See Also	Lists related	instructions.



#### 4.2 Instructions

The instructions are listed alphabetically, preceded by a summary.

#### Table 4-2. Summary of Instructions

#### Title

÷		ιu	ge
	ABSF32 RaH, RbH 32-bit Floating-Point Absolute Value	. 34	4
	ADDF32 RaH, #16FHi, RbH 32-bit Floating-Point Addition		
	ADDF32 RaH, RbH, #16FHi 32-bit Floating-Point Addition		
	ADDF32 RaH, RbH, RcH 32-bit Floating-Point Addition		
	ADDF32 RdH, ReH, RfH MOV32 mem32, RaH 32-bit Floating-Point Addition with Parallel Move		
	ADDF32 RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Addition with Parallel Move		
	CMPF32 RaH, RbH 32-bit Floating-Point Compare for Equal, Less Than or Greater Than		
	CMPF32 RaH, #16FHi 32-bit Floating-Point Compare for Equal, Less Than or Greater Than	. 4	5
	CMPF32 RaH, #0.0 32-bit Floating-Point Compare for Equal, Less Than or Greater Than	. 4	6
	EINVF32 RaH, RbH 32-bit Floating-Point Reciprocal Approximation	. 4	7
	EISQRTF32 RaH, RbH 32-bit Floating-Point Square-Root Reciprocal Approximation	. 4	9
	F32TOI16 RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Integer	. 5	1
	F32TOI16R RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Integer and Round	. 5	2
	F32TOI32 RaH, RbH Convert 32-bit Floating-Point Value to 32-bit Integer	. 5	3
	F32TOUI16 RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer	. 5	4
	F32TOUI16R RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer and Round	. 5	5
	F32TOUI32 RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer	. 5	6
	FRACF32 RaH, RbH Fractional Portion of a 32-bit Floating-Point Value	. 5	7
	I16TOF32 RaH, RbH Convert 16-bit Integer to 32-bit Floating-Point Value		
	I16TOF32 RaH, mem16 Convert 16-bit Integer to 32-bit Floating-Point Value		
	I32TOF32 RaH, mem32 Convert 32-bit Integer to 32-bit Floating-Point Value		
	I32TOF32 RaH, RbH Convert 32-bit Integer to 32-bit Floating-Point Value		
	MACF32 R3H, R2H, RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Add		2
	MACF32 R3H, R2H, RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Multiply and Accumulate with Paral Move		4
	MACF32 R7H, R3H, mem32, *XAR7++ 32-bit Floating-Point Multiply and Accumulate	. 6	6
	MACF32 R7H, R6H, RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Add	. 6	8
	MACF32 R7H, R6H, RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Multiply and Accumulate with Paral Move.		0
	MAXF32 RaH, RbH 32-bit Floating-Point Maximum		
	MAXF32 RaH, #16FHi 32-bit Floating-Point Maximum		
	MAXF32 RaH, RbH MOV32 RcH, RdH 32-bit Floating-Point Maximum with Parallel Move		
	MINF32 RaH, RbH 32-bit Floating-Point Minimum		
	MINF32 RaH, #16FHi 32-bit Floating-Point Minimum		
	MINF32 RaH, RbH MOV32 RcH, RdH 32-bit Floating-Point Minimum with Parallel Move		
	MOV16 mem16, RaH Move 16-bit Floating-Point Register Contents to Memory		
	MOV32 *(0:16bitAddr), loc32 Move the Contents of loc32 to Memory		
	MOV32 ACC, RaH Move 32-bit Floating-Point Register Contents to ACC		
	MOV32 loc32, *(0:16bitAddr) Move 32-bit Value from Memory to loc32	. 8	1
	MOV32 mem32, RaH Move 32-bit Floating-Point Register Contents to Memory		
	MOV32 mem32, STF Move 32-bit STF Register to Memory	. 8	3
	MOV32 P, RaH Move 32-bit Floating-Point Register Contents to P		
	MOV32 RaH, ACC Move the Contents of ACC to a 32-bit Floating-Point Register	. 8	5
	MOV32 RaH, mem32 {, CNDF} Conditional 32-bit Move		
	MOV32 RaH, P Move the Contents of P to a 32-bit Floating-Point Register	- 8	8



#### Table 4-2. Summary of Instructions (continued)

MOV32 RaH, RbH {, CNDF} Conditional 32-bit Move
MOV32 RaH, XARn Move the Contents of XARn to a 32-bit Floating-Point Register
MOV32 RaH, XT Move the Contents of XT to a 32-bit Floating-Point Register
MOV32 STF, mem32 Move 32-bit Value from Memory to the STF Register
MOV32 XARn, RaH Move 32-bit Floating-Point Register Contents to XARn
MOV32 XT, RaH Move 32-bit Floating-Point Register Contents to XT
MOVD32 RaH, mem32 Move 32-bit Value from Memory with Data Copy
MOVF32 RaH, #32F Load the 32-bits of a 32-bit Floating-Point Register
MOVI32 RaH, #32FHex Load the 32-bits of a 32-bit Floating-Point Register with the immediate
MOVIZ RaH, #16FHiHex Load the Upper 16-bits of a 32-bit Floating-Point Register
MOVIZF32 RaH, #16FHi Load the Upper 16-bits of a 32-bit Floating-Point Register
MOVST0 FLAG Load Selected STF Flags into ST0 100
MOVXI RaH, #16FLoHex Move Immediate to the Low 16-bits of a Floating-Point Register
MPYF32 RaH, RbH, RcH 32-bit Floating-Point Multiply 102
MPYF32 RaH, #16FHi, RbH 32-bit Floating-Point Multiply 103
MPYF32 RaH, RbH, #16FHi 32-bit Floating-Point Multiply 104
MPYF32 RaH, RbH, RcH ADDF32 RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Add 105
MPYF32 RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Multiply with Parallel Move 107
MPYF32 RdH, ReH, RfH MOV32 mem32, RaH 32-bit Floating-Point Multiply with Parallel Move 109
MPYF32 RaH, RbH, RcH ISUBF32 RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Subtract 110
NEGF32 RaH, RbH{, CNDF} Conditional Negation 111
POP RB Pop the RB Register from the Stack 112
PUSH RB Push the RB Register onto the Stack 113
RESTORE Restore the Floating-Point Registers 114
RPTB label, loc16 Repeat A Block of Code 116
RPTB label, #RC Repeat a Block of Code 118
SAVE FLAG, VALUE Save Register Set to Shadow Registers and Execute SETFLG 120
SETFLG FLAG, VALUE Set or clear selected floating-point status flags
SUBF32 RaH, RbH, RcH 32-bit Floating-Point Subtraction 123
SUBF32 RaH, #16FHi, RbH 32-bit Floating Point Subtraction
SUBF32 RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Subtraction with Parallel Move 125
SUBF32 RdH, ReH, RfH MOV32 mem32, RaH 32-bit Floating-Point Subtraction with Parallel Move 127
SWAPF RaH, RbH{, CNDF} Conditional Swap 128
TESTTF CNDF Test STF Register Flag Condition 129
UI16TOF32 RaH, mem16 Convert unsigned 16-bit integer to 32-bit floating-point value
UI16TOF32 RaH, RbH Convert unsigned 16-bit integer to 32-bit floating-point value
UI32TOF32 RaH, mem32 Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value
UI32TOF32 RaH, RbH Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value
ZERO RaH Zero the Floating-Point Register RaH 134
ZEROA Zero All Floating-Point Registers



Instructions

#### ABSF32 RaH, RbH 32-bit Floating-Point Absolute Value

#### Operands

Operands											
	RaHfloating-point destination register (R0H to R7H)RbHfloating-point source register (R0H to R7H)										
Opcode	LSW: 1110 01 MSW: 0000 00										
Description	The absolute value of RbH is loaded into RaH. Only the sign bit of the operand is modified by the ABSF32 instruction.										
	if (RbH < 0 else	) {RaH = -RbH] {RaH = RbH]									
Flags	This instruction modifies the following flags in the STF register:										
	Flag TF	ZI	NI	ZF	NF	LUF	LVF				
	Modified No	No	No	Yes	Yes	No	No				
	NF = 0; ZF = 0;	ister flags are		s follows:							
Pipeline	This is a sing	gle-cycle instru	uction.								
Example	MOVIZF32 ABSF32	R1H, #-2.0 R1H, R1H		2.0 (0xC000 2.0 (0x4000	,	NF = 0					
	MOVIZF32 ABSF32	ROH, #5.0 ROH, ROH		5.0 (0x40A0 5.0 (0x40A0	,	NF = 0					
	MOVIZF32 ABSF32	R0H, #0.0 R1H, R0H		0.0 0.0 ZF = 1	, NF = 0						
See also	NEGF32 Ra	H, RbH{, CND	F}								



#### ADDF32 RaH, #16FHi, RbH 32-bit Floating-Point Addition

#### Operands

Operands											
	RaH	floating-point destination register (R0H to R7H)									
	#16FHi A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-I floating-point value. The low 16-bits of the mantissa are assumed to be all										
	RbH										
Opcode	LSW: 1110 1000 10II IIII MSW: IIII IIII IIbb baaa										
Description	Add RbH to the floating-point value represented by the immediate operand. Store the result of the addition in RaH.										
	<ul> <li>#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.</li> <li>RaH = RbH + #16FHi:0</li> <li>This instruction can also be written as ADDF32 RaH, RbH, #16FHi.</li> </ul>										
Flags	This instruction r	nodifies the f	following fla	igs in the S	TF register	r:					
	Flag TF	ZI	NI	ZF	NF	LUF	LVF				
	Modified No	No	No	No	No	Yes	Yes				
Pipeline	<ul> <li>The STF register flags are modified as follows:</li> <li>LUF = 1 if ADDF32 generates an underflow condition.</li> <li>LVF = 1 if ADDF32 generates an overflow condition.</li> <li>This is a 2 pipeline-cycle instruction (2p). That is: ADDF32 RaH, #16FHi, RbH ; 2 pipeline cycles (2p)</li> </ul>										
	NOP ; 1 cycle delay or non-conflicting instruction ; < ADDF32 completes, RaH updated										
	Any instruction in the delay slot must not use RaH as a destination register or use RaH as a source operand.										
Example	kample; Add to RlH the value 2.0 in 32-bit floating-point f ADDF32 R0H, #2.0, RlH; R0H = 2.0 + RlH ; Delay for ADDF32 to com ; < ADDF32 completes, R ; NOP										
	; Add to R3H th ADDF32 R2H, NOP NOP		in 32-bit floating-point format ; R2H = -2.5 + R3H ; Delay for ADDF32 to complete ; < ADDF32 completes, R2H updated ;								
	; Add to R5H th ADDF32 R5H, NOP NOP		C00000 (1.5) ; R5H = 1.5 + R5H ; Delay for ADDF32 to complete ; < ADDF32 completes, R5H updated ;								



See also

ADDF32 RaH, RbH, #16FHi ADDF32 RaH, RbH, RcH ADDF32 RdH, ReH, RfH || MOV32 RaH, mem32 ADDF32 RdH, ReH, RfH || MOV32 mem32, RaH MACF32 R3H, R2H, RdH, ReH, RfH MPYF32 RaH, RbH, RcH || ADDF32 RdH, ReH, RfH



# ADDF32 RaH, RbH, #16FHi 32-bit Floating-Point Addition

Operands									
	RaH	floating-poir	nt destination	register (R0H t	o R7H)				
	RbH	floating-poir	nt source regi	ster (R0H to R	7H)				
	#16FHi			that represent low 16-bits of t					
Opcode	LSW: 1110 1000 10II IIII MSW: IIII IIII IIbb baaa								
Description	Add RbH to the result of the add			presented by	/ the immed	liate operan	d. Store the		
	#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x4000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.								
	This instruction	can also be	written as	ADDF32 Ra	aH, #16FHi,	RbH.			
Flags	This instruction	modifies the	following	flags in the	STF registe	r:			
	Flag TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified No	No	No	No	No	Yes	Yes		
Pipeline	<ul> <li>The STF register flags are modified as follows:</li> <li>LUF = 1 if ADDF32 generates an underflow condition.</li> <li>LVF = 1 if ADDF32 generates an overflow condition.</li> <li>This is a 2 pipeline-cycle instruction (2p). That is: <ul> <li>ADDF32 RaH, #16FHi, RbH</li> <li>i 2 pipeline cycles (2p)</li> <li>i 1 cycle delay or non-conflicting instruction</li> <li>i - ADDF32 completes, RaH updated</li> </ul> </li> </ul>								
	NOP Any instruction i as a source ope		slot must r	not use RaH	as a destin	ation registe	er or use RaH		
Example	; Add to RlH th	; Add to R1H the value 2.0 in 32-bit floating-point format ADDF32 R0H, R1H, #2.0 ; R0H = R1H + 2.0 NOP ; Delay for ADDF32 to complete ; < ADDF32 completes, R0H updated							
	; Add to R3H th ADDF32 R2H, NOP		; R2H = ; Delay ; < 7	it floating = R3H + (-2 y for ADDF32 ADDF32 comp	.5) 2 to comple	te			
	NOP		;						
	; Add to R5H th ADDF32 R5H, NOP		20 ; R5H ; ; Delay						
	NOP		;						



See also

ADDF32 RaH, #16FHi, RbH ADDF32 RaH, RbH, RcH ADDF32 RdH, ReH, RfH || MOV32 RaH, mem32 ADDF32 RdH, ReH, RfH || MOV32 mem32, RaH MACF32 R3H, R2H, RdH, ReH, RfH MPYF32 RaH, RbH, RcH || ADDF32 RdH, ReH, RfH



# ADDF32 RaH, RbH, RcH 32-bit Floating-Point Addition

Operands								
	RaH	floa	ting-poi	nt destination	register (R0H	to R7H)		
	RbH	floa	ting-poi	nt source regi	ster (R0H to R	.7H)		
	RcH		• ·	•	ster (R0H to R	,		
			01	0	,	,		
Opcode	LSW: 111 MSW: 000	.0 0111 0001 00 000c ccbb	0000 baaa					
Description	Add the contents of RcH to the contents of RbH and load the result into RaH. RaH = $RbH$ + $RcH$							
Flags	This inst	ruction modif	ies the	e following	flags in the	STF registe	er:	
	Flag	TF ZI		NI	ZF	NF	LUF	LVF
	Modified	No No	)	No	No	No	Yes	Yes
	<ul><li>LUF</li><li>LVF</li></ul>	Fregister flag = 1 if ADDF3 = 1 if ADDF3	2 gene 2 gene	erates an u erates an o	nderflow co verflow con			
Pipeline		2 pipeline-cy		`	. ,			
	ADDF3 NOP	32 RaH, RbH,	RCH		peline cycl cle delay o		licting inst	cruction
	NOD			-	ADDF32 comp			
	NOP			_				
	-	ruction in the irce operand.	delay	slot must r	iot use RaH	l as a destir	nation regist	er or use RaH
Example		e Y = M1*X1 ata page.	+ B1.	This exam	ole assume	s that M1, >	۲, B1 and ۱)	r are all on the
	MOVW MOV32 MOV32 MPYF3    MOV32 NOP	2 R1H,@X1 32 R1H,R1H,R 2 R0H,@B1	; ; :OH ; ; ;	<pre> MOV32 </pre>	with M1 with X1 M1*X1 complete 2 complete			
	ADDF3 NOP	32 R1H,R1H,R			to B1 and		LH	
	MOV32	2 @Y1,R1H		Store the	32 complete e result			
	Calculat	e Y = A + B.						
	MOVL MOV32	XAR4, #A ROH, *XAR XAR4, #B RIH, *XAR 2 ROH,RIH,RO	.4 ;	Load R0H v Load R1H v Add A + B		н		
		XAR4, #Y 2 *XAR4,ROH		< ADDF32 Store the	-			
See also	ADDF32 ADDF32 ADDF32 MACF32	2 RaH, #16FF 2 RaH, RbH, 3 2 RdH, ReH, 1 2 RdH, ReH, 1 2 R3H, R2H, 1 2 RaH, RbH, 1	#16FH RfH      RfH      RdH, F	i MOV32 Ra MOV32 me ReH, RfH	m32, RaH	RfH		



# ADDF32 RdH, ReH, RfH MOV32 mem32, RaH 32-bit Floating-Point Addition with Parallel Move

Operands									
	RdH	floating-poi	nt destinatior	n register for the	ADDF32 (R0	H to R7H)			
	ReH	floating-poi	nt source reg	ister for the ADD	0F32 (R0H to	R7H)			
	RfH floating-point source register for the ADDF32 (R0H to R7H)								
	mem32	pointer to a	32-bit memo	ory location. This	will be the de	estination of the	e MOV32.		
	RaH	floating-poi	nt source reg	jister for the MO	V32 (R0H to I	R7H)			
Oncodo	T CHI. 1110 00	00 0001 555-							
Opcode	LSW: 1110 00 MSW: eedd da								
Description	the result in F by mem32. n modes suppo RdH = ReH +	Perform an ADDF32 and a MOV32 in parallel. Add RfH to the contents of ReH and store the result in RdH. In parallel move the contents of RaH to the 32-bit location pointed to by mem32. mem32 addresses memory using any of the direct or indirect addressing modes supported by the C28x CPU. RdH = ReH + RfH, [mem32] = RaH							
Flags	This instruction	on modifies the	e following	flags in the S	STF registe	r:			
	Flag TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified No	No	No	No	No	Yes	Yes		
Pipeline	ADDF32 is a ADDF32    MOV32 NOP	MOV32 mem32, RaH ; 1 cycle ; < MOV32 completes, mem32 updated							
	NOP								
	Any instruction as a source of	on in the delay operand.	slot must	not use RdH	as a destir	ation regist	er or use RdH		
Example	ADDF32    MOV32	R3H, R6H, H R7H, *-SP[2		; (A) R3; ; ; < R7		R4H and R7H	I = I3		
	SUBF32	R6H, R6H, 1	R4H		H = R6H - DF32 (A) c	R4H completes, F	3H valid		
	SUBF32    MOV32	R3H, R1H, I *+XAR5[2],		; (C) R3	H = R1H -	R7H and sto	ore R3H (A)		
		TARCE [2],	KJII			ompletes, R tes, (A) st			
	ADDF32	R4H, R7H, 1	R1H		-	R1H and sto			
	MOV32	*+XAR5[6],	R6H	; ; < SU	BF32 (C) c	completes, R	3H valid		
	MOV32	*+XAR5[0],	R3H	; store ; < MO	R3H (C) V32 comple	tes, (B) st tes, (C) st completes, F	ored		
	MOV32	*+XAR5[4],	R4H	; store	R4H (D)	etes, (D) st			



www.ti.com		Instructions
See also	ADDF32 RaH, #16FHi, RbH ADDF32 RaH, RbH, #16FHi ADDF32 RaH, RbH, RcH MACF32 R3H, R2H, RdH, ReH, RfH MPYF32 RaH, RbH, RcH    ADDF32 RdH, ReH, RfH ADDF32 RdH, ReH, RfH    MOV32 RaH, mem32	

# ADDF32 RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Addition with Parallel Move

Operands									
	RdH				register for the register as Ra		H to R7H).		
	ReH		floating-po	int source regi	ster for the AD	DF32 (R0H to	R7H)		
	RfH		R7H)						
	RaH				register for the register as Ro		I to R7H).		
	mem32		pointer to a	a 32-bit memo	ry location. Thi	s is the source	e for the MOV3	2.	
Opcode	LSW: 1110 0011 0001 fffe MSW: eedd daaa mem32								
Description	Perform an ADDF32 and a MOV32 operation in parallel. Add RfH to the contents of ReH and store the result in RdH. In parallel move the contents of the 32-bit location pointed to by mem32 to RaH. mem32 addresses memory using any of the direct or indirect addressing modes supported by the C28x CPU. RdH = ReH + RfH, RaH = [mem32]								
Restrictions				the ADDF3		IOV32 mus	t be unique	. That is, RaH	
	Any insti as a sou		•	slot must r	not use RdH	as a destir	ation regist	er or use RdH	
Flags	This instruction modifies the following flags in the STF register:								
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	Yes	Yes	Yes	Yes	Yes	Yes	
	<ul> <li>The STF register flags are modified as follows:</li> <li>LUF = 1 if ADDF32 generates an underflow condition.</li> <li>LVF = 1 if ADDF32 generates an overflow condition.</li> <li>The MOV32 Instruction will set the NF, ZF, NI and ZI flags as follows:</li> <li>NF = RaH(31); ZF = 0; if(RaH(30:23) == 0) { ZF = 1; NF = 0; }</li> </ul>								
	The MO' NF = RaH ZF = 0; if(RaH(3 NI = RaH	V32 Inst (31); 0:23) ==	DF32 gen ruction will	erates an o set the NF	verflow con , ZF, NI anc	dition.	follows:		
	The MO' NF = RaH ZF = 0; if(RaH(3 NI = RaH ZI = 0;	V32 Inst (31); (32) == (31);	DF32 gen ruction will	erates an o set the NF = 1; NF = (	verflow con , ZF, NI anc	dition.	follows:		
Pipeline	The MO' NF = RaH ZF = 0; if(RaH(3 NI = RaH ZI = 0; if(RaH(3	V32 Inst ((31); ((31); ((31); ((31); 1:0) ==	DF32 gen ruction will = 0) { ZF 0) ZI = 1	erates an o set the NF = 1; NF = ( ;	verflow con , ZF, NI anc	dition. I ZI flags as		cycle. That is:	
Pipeline	The MO' NF = RaH ZF = 0; if(RaH(3 NI = RaH ZI = 0; if(RaH(3	V32 Inst ((31); ((31); ((31); (1:0) == DF32 tak (32) RdH	DF32 gen ruction will = 0) { ZF 0) ZI = 1	erates an o set the NF = 1; NF = ( ; ne cycles (; ; 2 pipe ; 1 cycl	verflow con , ZF, NI and ); } 2p) and the eline cycle	dition. I ZI flags as MOV32 tak s (2p)	es a single	cycle. That is:	
Pipeline	The MO' NF = RaH ZF = 0; if(RaH(3 NI = RaH ZI = 0; if(RaH(3 The ADE	V32 Inst ((31); ((31); ((31); (1:0) == DF32 tak (32) RdH	DF32 gen ruction will = 0) { ZF 0) ZI = 1 res 2 pipeli , ReH, RfH	erates an o set the NF = 1; NF = ( ; ne cycles (; ; 2 pipe ; 1 cyc; ; 3 cyc; ; 1 cyc; ; 1 cyc;	verflow con , ZF, NI and ); } 2p) and the eline cycle le DV32 comple	dition. I ZI flags as MOV32 tak s (2p) tes, RaH up non-confli	<b>es a single</b> dated cting instr		



www.ti.com	
Example	Calculate $Y = A + B - C$ :
	MOVL XAR4, #A
	MOV32 ROH, *XAR4 ; Load ROH with A
	MOVL XAR4, #B
	MOV32 R1H, *XAR4 ; Load R1H with B
	MOVL XAR4, #C
	ADDF32 R0H,R1H,R0H ; Add A + B and in parallel
	MOV32 R2H, *XAR4 ; Load R2H with C
	; < MOV32 complete
	MOVL XAR4, #Y
	; ADDF32 complete
	SUBF32 ROH,ROH,R2H ; Subtract C from (A + B) NOP
	; < SUBF32 completes
	MOV32 *XAR4,ROH ; Store the result
See also	ADDF32 RaH, #16FHi, RbH
	ADDF32 RaH, RbH, #16FHi
	ADDF32 RaH, RbH, RcH
	ADDF32 RdH, ReH, RfH    MOV32 mem32, RaH
	MACF32 R3H, R2H, RdH, ReH, RfH
	MPYF32 RaH, RbH, RcH    ADDF32 RdH, ReH, RfH
	$\lambda = 0$ . The set of $0$

Instructions

TEXAS INSTRUMENTS

www.ti.com

# CMPF32 RaH, RbH 32-bit Floating-Point Compare for Equal, Less Than or Greater Than

••••••••••••••••			<b>1</b>	, <u></u>					
Operands									
	RaH	floating-poi	nt source reg	ister (R0H to R	7H)				
	RbH	floating-poi	nt source reg	ster (R0H to R	7H)				
Opcode		LSW: 1110 0110 1001 0100 MSW: 0000 0000 00bb baaa							
Description	as a logical	NF flags on the compare operation of the ot. Basically the	tion. This is	s possible be	ecause of the	ne IEEE forr	nat offsetting		
	Special case	es for inputs:							
	•	zero will be tre	ated as po	sitive zero.					
	-	nalized value w	•		e zero.				
		umber (NaN) wil		•					
Flags		tion modifies the			STF registe	er:			
	Flag TF		NI	ZF	NF	LUF	LVF		
	Modified No	No	No	Yes	Yes	No	No		
Pipeline Example	<pre>If(RaH &lt; RbH) {ZF=0, NF=1} This is a single-cycle instruction. ; Behavior of ZF and NF flags for different comparisons MOVIZF32 R1H, #-2.0 ; R1H = -2.0 (0xC0000000) MOVIZF32 R0H, #5.0 ; R0H = 5.0 (0x40A00000)</pre>								
	CMPF32	ROH, #5.0 R1H, ROH	; $ZF = 0$		,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,				
	CMPF32         R0H, R1H         ; ZF = 0, NF = 0           CMPF32         R0H, R0H         ; ZF = 1, NF = 0								
	; Using the result of a compare for loop control								
	Loop: MOV32 MOV32 CMPF32 MOVST0 BF	R0H,*XAR4++ R1H,*XAR3++ R1H, R0H ZF, NF Loop, GT	; Copy Z			N bits			
See also	CMPF32 Ra CMPF32 Ra MAXF32 Ra MAXF32 Ra MINF32 Ra MINF32 Ra	aH, #0.0 aH, #16FHi aH, RbH H, #16FHi							



# CMPF32 RaH, #16FHi 32-bit Floating-Point Compare for Equal, Less Than or Greater Than

Operands										
	RaH	floating-poi	int source regis	ster (R0H to R7	H)					
	#16FHi			that represents ow 16-bits of th						
Opcode	LSW: 1110 100 MSW: IIII III									
Description	Compare the operand. Set					ented by th	e immediate			
	#16FHi is a 1 floating-point addressing m are 0. Some e -1.5 (0xBFC0 value. That is	value. The low ode is most us examples are 0000). The as	w 16-bits of seful for con 2.0 (0x4000 sembler wil	the mantissanstants when 00000), 4.0 (0 I accept eithe	a are assur e the lowes 0x4080000 er a hex or	ned to be a st 16-bits of 0), 0.5 (0x3	ll 0. This the mantissa F000000), and			
	because of th	The CMPF32 instruction is performed as a logical compare operation. This is possible because of the IEEE floating-point format offsets the exponent. Basically the bigger the binary number, the bigger the floating-point value.								
	Special cases	Special cases for inputs:								
	Denormali	zero will be tre zed value will nber (NaN) wi	be treated	as positive z	ero.					
Flags	This instruction	on modifies the	e following f	lags in the S	TF registe	r:				
	Flag TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified No	No	No	Yes	Yes	No	No			
	The STF regis If(RaH == #16 If(RaH > #16 If(RaH < #16	5FHi:0) {ZF=1 5FHi:0) {ZF=0	, NF=0}, NF=0}	follows:						
Pipeline	This is a sing	le-cycle instru	ction							
Example	; Behavior of	ZF and NF f	lags for di	fferent com	parisons					
	MOVIZF32R1H, #-2.0; R1H = -2.0 (0xC000000)MOVIZF32R0H, #5.0; R0H = 5.0 (0x40A00000)CMPF32R1H, #-2.2; ZF = 0, NF = 0CMPF32R0H, #6.5; ZF = 0, NF = 1CMPF32R0H, #5.0; ZF = 1, NF = 0									
	; Using the result of a compare for loop control									
	Loop: MOV32 CMPF32 MOVST0 BF	R1H,*XAR3++ R1H, #2.0 ZF, NF Loop, GT	; Copy ZF	H ar ZF and NI and NF to S R1H > #2.0		N bits				
See also	CMPF32 RaH CMPF32 RaH MAXF32 RaH MAXF32 RaH MINF32 RaH, MINF32 RaH,	I, RbH I, #16FHi I, RbH , #16FHi								

TEXAS INSTRUMENTS

#### Operands RaH floating-point source register (R0H to R7H) #0.0 zero Opcode LSW: 1110 0101 1010 0aaa Description Set the ZF and NF flags on (RaH - #0.0). The CMPF32 instruction is performed as a logical compare operation. This is possible because of the IEEE floating-point format offsets the exponent. Basically the bigger the binary number, the bigger the floating-point value. Special cases for inputs: Negative zero will be treated as positive zero. • Denormalized value will be treated as positive zero. Not-a-Number (NaN) will be treated as infinity. • Flags This instruction modifies the following flags in the STF register: Flag TF ΖI NI ZF NF LUF LVF Modified No No No Yes Yes No No The STF register flags are modified as follows: If(RaH == #0.0) {ZF=1, NF=0} If(RaH > #0.0) {ZF=0, NF=0} $If(RaH < \#0.0) \{ZF=0, NF=1\}$ **Pipeline** This is a single-cycle instruction. Example ; Behavior of ZF and NF flags for different comparisons MOVIZF32 R0H, #5.0 ; $ROH = 5.0 (0 \times 40 \land 0000)$ MOVIZF32 R1H, #-2.0 ; R1H = -2.0 (0xC000000)MOVIZF32 R2H, #0.0 ; R2H = 0.0 (0x0000000); ZF = 0, NF = 0ROH, #0.0 CMPF32 CMPF32 R1H, #0.0 ; ZF = 0, NF = 1R2H, #0.0 ; ZF = 1, NF = 0CMPF32 ; Using the result of a compare for loop control Loop: ; Load R1H R1H,\*XAR3++ MOV32 CMPF32 R1H, #0.0 ; Set/clear ZF and NF MOVST0 ZF, NF ; Copy ZF and NF to STO Z and N bits BF Loop, GT ; Loop if R1H > #0.0 CMPF32 RaH, #0.0 See also CMPF32 RaH, #16FHi MAXF32 RaH, #16FHi MAXF32 RaH, RbH MINF32 RaH, #16FHi MINF32 RaH, RbH

#### CMPF32 RaH, #0.0 32-bit Floating-Point Compare for Equal, Less Than or Greater Than



# EINVF32 RaH, RbH 32-bit Floating-Point Reciprocal Approximation

Operands									
	RaH		floating-p	oint destinatior	register (R0H	to R7H)			
	RbH		floating-p	oint source reg	ister (R0H to F	R7H)			
Opcode	LSW: 111 MSW: 000		1001 001 00bb baa						
Description	approxir more ac Ye = Est	nately 8 curate a	bits. This answer. Th	value can b				accurate to thm to get a	
	$Ye = Ye^{2}$ $Ye = Ye^{2}$								
	10 10	(2.0	ic n,						
	accurate approxir DeNorm	e to the 3 nately d i or NaN	32-bit float oubles. Th I value.	ing-point for	rmat. Õn ea	n, you will g ch iteration t vill not gener	the mantissa	a bit accuracy	
	RaH = Es	stimate	of 1/RbH						
Flags	This ins	truction	modifies th	ne following	flags in the	STF registe	r:		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	Yes	Yes	
Pipeline									I



1130 000013		www.u.com
Example	Calculate Y = A/B. A fast divis C28x FPU Fast RTS Library (	sion routine similar to that shown below can be found in the SPRC664).
	MOVL XAR4, #A	
	MOV32 R0H, *XAR4 ; MOVL XAR4, #B	Load ROH with A
	,	Load R1H with B
		Calculate ROH = ROH / R1H
	MOV32 *XAR4, R0H ;	
	DIV:	
	EINVF32 R2H, R1H	; $R2H = Ye = Estimate(1/B)$
	CMPF32 R0H, #0.0	; Check if $A == 0$
	MPYF32 R3H, R2H, R1	1H ; R3H = Ye*B
	NOP	
	SUBF32 R3H, #2.0, F NOP	R3H ; R3H = 2.0 - Ye*B
	MPYF32 R2H, R2H, R3 NOP	3H ; R2H = Ye = Ye*(2.0 - Ye*B)
	MPYF32 R3H, R2H, R1	1H ; R3H = Ye*B
	CMPF32 R1H, #0.0	
	SUBF32 R3H, #2.0, F	
	NEGF32 ROH, ROH, EC	
	MPYF32 R2H, R2H, R3 NOP	
	-	2H ; ROH = Y = A*Ye = A/B

See also

EISQRTF32 RaH, RbH



# EISQRTF32 RaH, RbH 32-bit Floating-Point Square-Root Reciprocal Approximation

RaH		floating-po	oint destination	register (R0H	to R7H)		
RbH		floating-po	oint source reg	ister (R0H to F	27H)		
to appro	ximately	/ 8 bits. Th	is value car				
Ye = Ye	*(1.5 -	Ye*Ye*X/2	.0)				
accurate approxir DeNorm	e to the nately d or NaN	32-bit float oubles. Th I value.	ing-point for e EISQRTF	mat. Õn ea	ch iteration	the mantissa	a bit accuracy
This ins	truction	modifies th	e following	flags in the	STF registe	er:	
Flag	TF	ZI	NI	ZF	NF	LUF	LVF
Modified	No	No	No	No	No	Yes	Yes
LUF     LVF     This is a     EINN     NOP     NOP     Any inst	= 1 if EI = 1 if EI a 2 pipel /F32 Ra	SQRTF32 SQRTF32 ine cycle (ź , ĸbн n the delay	generates generates 2p) instructi ; 2 pip ; 1 cyc ; < E	an underflow an overflow on. That is: eline cycle le delay or ISQRTF32 cc	condition. s (2p) non-confli mpletes, Ra	H updated	
	RbHLSW: 111MSW: 000This operationto approximate ap	RbHLSW: 1110 0110 MSW: 0000 0000This operation g to approximately more accurate a Ye = Stimate(1 Ye = Ye*(1.5 - Ye = Ye*(1.5 - Ye = Ye*(1.5 - Ye = Ye*(1.5 -After 2 iterations accurate to the a approximately d DeNorm or NaN RaH = EstimateThis instructionFlagTF Modified NoThe STF register • LUF = 1 if EI This is a 2 pipel EINVF32 NOP NOPAny instruction i	RbHfloating-pcLSW: 1110 0110 1001 0010MSW: 0000 0000 00bb baasThis operation generates a to approximately 8 bits. Th more accurate answer. This Ye = Setimate(1/sqrt(X)); Ye = Ye*(1.5 - Ye*Ye*X/2)Ye = Estimate(1/sqrt(X)); Ye = Ye*(1.5 - Ye*Ye*X/2)After 2 iterations of the Net accurate to the 32-bit floati approximately doubles. Th DeNorm or NaN value.RaH = Estimate of 1/sqrtThis instruction modifies thFlagTFZIModifiedNoNOPNOP	RbHfloating-point source regLSW: 1110 0110 1001 0010 MSW: 0000 0000 00bb baaaThis operation generates an estimate to approximately 8 bits. This value car more accurate answer. That is:Ye = Estimate(1/sqrt(X)); Ye = Ye*(1.5 - Ye*Ye*X/2.0) Ye = Ye*(1.5 - Ye*Ye*X/2.0)After 2 iterations of the Newton-Raphs accurate to the 32-bit floating-point for approximately doubles. The EISQRTF DeNorm or NaN value. RaH = Estimate of 1/sqrt (RbH)This instruction modifies the followingFlagTFZlNoNoNoNoNoNoNoEINVF32RaH, RbH $(2 pip)$ NOPNOP $(1 cyc)$ $(< - E)$ Any instruction in the delay slot must of	RbHfloating-point source register (R0H to RLSW: 1110 0110 1001 0010 MSW: 0000 0000 00bb baaaThis operation generates an estimate of 1/sqrt(X) to approximately 8 bits. This value can be used in more accurate answer. That is:Ye = Estimate(1/sqrt(X)); Ye = Ye*(1.5 - Ye*Ye*X/2.0) Ye = Ye*(1.5 - Ye*Ye*X/2.0)After 2 iterations of the Newton-Raphson algorithm accurate to the 32-bit floating-point format. On ear approximately doubles. The EISQRTF32 operation DeNorm or NaN value.RaH = Estimate of 1/sqrt (RbH)This instruction modifies the following flags in theFlagTFZINIZFModifiedNoNoNoNoThe STF register flags are modified as follows:LUF = 1 if EISQRTF32 generates an underflowLUF = 1 if EISQRTF32 generates an overflowThis is a 2 pipeline cycle (2p) instruction. That is: EINVF32 RaH, RbH NOPEINVF32 RaH, RbH S of the cycle delay or S of the cycle delay or S of the cycle delay of the cycle delay of the cycle delay of the cycle delay store	RbHfloating-point source register (R0H to R7H)LSW: 1110 0110 1001 0010 MSW: 0000 0000 00bb baaaImage: Source register (R0H to R7H)LSW: 1110 0110 1001 0010 MSW: 0000 0000 00bb baaaThis operation generates an estimate of 1/sqrt(X) in 32-bit float to approximately 8 bits. This value can be used in a Newton-I more accurate answer. That is: Ye = Estimate(1/sqrt(X)); Ye = Ye*(1.5 - Ye*Ye*X/2.0) Ye = Ye*(1.5 - Ye*Ye*X/2.0)After 2 iterations of the Newton-Raphson algorithm, you will g accurate to the 32-bit floating-point format. On each iteration approximately doubles. The EISQRTF32 operation will not ge DeNorm or NaN value. RaH = Estimate of 1/sqrt (RbH)This instruction modifies the following flags in the STF registerFlagTFZINIZFNFModifiedNoNoNoNoNoNoNoNoNoNoNoNoSuppeline cycles (2p) (1 cycle delay or non-confli (2 confletes, Ra NOPAny instruction in the delay slot must not use RaH as a destif	RbHfloating-point source register (R0H to R7H)LSW: 1110 0110 1001 0010 MSW: 0000 0000 00bb baaaDiscrete register (R0H to R7H)This operation generates an estimate of 1/sqrt(X) in 32-bit floating-point for to approximately 8 bits. This value can be used in a Newton-Raphson alg more accurate answer. That is: $Ye = Estimate(1/sqrt(X));$ $Ye = Ye*(1.5 - Ye*Ye*X/2.0)$ $Ye = Ye*(1.5 - Ye*Ye*X/2.0)$ After 2 iterations of the Newton-Raphson algorithm, you will get an exact accurate to the 32-bit floating-point format. On each iteration the mantissa approximately doubles. The EISQRTF32 operation will not generate a neg DeNorm or NaN value. RaH = Estimate of 1/sqrt (RbH)This instruction modifies the following flags in the STF register: <a href="#">Image TF</a> <a href="#">ZI</a> NoPAfter 2 iterations of the Stretce and overflow condition.LUF = 1 if EISQRTF32 generates an underflow condition. <a href="#">LUF = 1 if EISQRTF32 generates an overflow condition.</a> <a href="#">LUF = 1 if EISQRTF32 generates an overflow condition.</a> <a href="#">LUF = 1 if EISQRTF32 generates an overflow condition.</a> <a href="#">LUF = 1 if EISQRTF32 generates an overflow condition.</a> <a href="#">LUF = 1 if EISQRTF32 generates an overflow condition.</a> <a href="#">MOP</a> <



Instructions

Example

Calculate the square root of X. A square-root routine similar to that shown below can be found in the *C28x FPU Fast RTS Library* (<u>SPRC664</u>).

; Ye = Ye*(1.5 ; Y = X*Ye	- Ye*Ye*X*0.5)	
_sqrt:		
		; ROH = X on entry
EISQRTF32	R1H, R0H	; R1H = Ye = Estimate(1/sqrt(X))
MPYF32	R2H, R0H, #0.5	; R2H = X*0.5
MPYF32 NOP	R3H, R1H, R1H	; R3H = Ye*Ye
MPYF32 NOP	R3H, R3H, R2H	; R3H = Ye*Ye*X*0.5
SUBF32 NOP	R3H, #1.5, R3H	; R3H = 1.5 - Ye*Ye*X*0.5
MPYF32 NOP	R1H, R1H, R3H	; R2H = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
MPYF32 NOP	R3H, R1H, R2H	; R3H = Ye*X*0.5
MPYF32 NOP	R3H, R1H, R3H	; R3H = Ye*Ye*X*0.5
SUBF32	R3H, #1.5, R3H	; R3H = 1.5 - Ye*Ye*X*0.5
CMPF32		; Check if $X == 0$
MPYF32 NOP		; R2H = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
MOV32		; If X is zero, change the Ye estimate to 0
MOV32 MPYF32 LRETR		; ROH = Y = $X*Ye$ = sqrt(X)

See also

EINVF32 RaH, RbH



# F32TOI16 RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Integer

Operands								
	RaH	floating-poir	nt destination	register (R0H	to R7H)			
	RbH	floating-poir	nt source regi	ster (R0H to R	7H)			
Opcode	LSW: 1110 0110 MSW: 0000 0000							
Description	Convert a 32-b will be stored in RaH(15:0) = F RaH(31:16) = s	n <b>RaH.</b> 32TOI16(RbH)			6-bit intege	r and trunca	e. The result	
Flags	This instruction	does not aff	ect any flag	gs:				
	Flag TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified No	No	No	No	No	No	No	
Pipeline Example	This is a 2 pipe F32TOI16 NOP NOP Any instruction as a source op MOVIZF32	п the delay	; 2 pir ; 1 cyo ; < 1 slot must r	eeline cycl le delay o: 32TOII6 con ot use RaH	r non-confl mpletes, Ra	_		ł
	F32TOI16 MOVIZF32 F32TOI16 NOP	R1H, R0H R2H, #-5.0 R3H, R2H	<pre>; R1H(31::; ; R2H = -5; ; &lt; F32; ; R3H(15:0; ; R3H(31::; ; 1 Cycle</pre>	COI16 comple ) = F32TO .6) = Sign delay for 2	extension c 0000) ete, R1H(15 R1H(31 I16(R2H) extension c F32TOI16 tc ete, R3H(15	5:0) = 5 (0 .:16) = 0 (0 of R3H(15)	x0000) 0xFFFB)	
See also	F32TOI16R Ra F32TOUI16 Ra F32TOUI16R F I16TOF32 RaH I16TOF32 RaH UI16TOF32 Ra UI16TOF32 Ra	H, RbH RaH, RbH , RbH , mem16 H, mem16						



# F32TOI16R RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Integer and Round

Operanus							
	RaH	floating-point of	destination	register (R0H	to R7H)		
	RbH	floating-point :	source regi	ster (R0H to F	R7H)		
					,		
Opcode	LSW: 1110 0110 MSW: 1000 0000						
Description	Convert the 32 even value. Th				16-bit integ	ger and roun	d to the nearest
	RaH(15:0) = F RaH(31:16) = s			.5)			
Flags	This instruction	does not affeo	t any flag	js:			
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No	No	No	No	No	No	No
Pipeline	This is a 2 pipe	line cycle (2p)	instructio	n. That is:			
	F32TOI16R NOP		; 2 pip ; 1 cyc	eline cycl le delay c	or non-conf	licting inst RaH updated	ruction
	NOP		/ < 1	JZIOIIOR C	Julipieces, 1	an updated	
	Any instruction as a source op		ot must n	ot use Ra⊦	l as a desti	nation regist	er or use RaH
Example	MOVIZ	ROH, #0x3FD9	; R0H [	31:16] = 0	x3FD9		
-	MOVXI	ROH, #0x999A		15:0] = 0			
	F32TOI16R	R1H, R0H	; R1H(1		2TOI16round		
	MOVF32	R2H, #-1.7		1:16) = Si -1.7 (0xE		on of R1H(15	;)
		, "			omplete, R	1H(15:0) =	
	F32TOI16R	R3H, R2H	; ; R3H(1	(5:0) = F3	R: 2TOI16round	1H(31:16) = d (R2H)	0 (0x0000)
			; R3H(3	1:16) = Si	gn extensio	on of R2H(15	
	NOP		-	-	omplete, R	6R to comple 1H(15:0) = 1H(31:16) =	-2 (OxFFFE)
See also	F32TOI16 RaH F32TOUI16 Ra F32TOUI16R F I16TOF32 RaH I16TOF32 RaH UI16TOF32 Ra UI16TOF32 Ra	H, RbH RaH, RbH I, RbH I, mem16 ∖H, mem16					



# F32TOI32 RaH, RbH Convert 32-bit Floating-Point Value to 32-bit Integer

Operands								
	RaH		floating-po	int destinatior	n register (R0H	I to R7H)		
	RbH		floating-po	int source reg	jister (R0H to F	R7H)		
Opcode	LSW: 11: MSW: 00		1000 1000 00bb baaa					
Description	Convert Store th	e result	in RaH.	point value	in RbH to a	a 32-bit integ	ger value an	d truncate.
Flags	This ins	truction	does not at	ffect any fla	ags:			
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
	NOP	truction i		; 1 cy ; <	F32T0I32 co	or non-confi ompletes, Ra	_	eruction
Example	MOVI	TOI32 F F32 F	R3H, R2H	; R 94005.0 ; R ; < ; R ; R ; 1 ; 2	3H = F32T01 4H = -11204 F32T0132 3H = 112040 5H = F32T01 Cycle dela F32T0132	4005.0 (0xCH 2 complete, 005 (0x00AAH 132 (R4H) ay for F32TC	32AF5A5) F5A5) DI32 to comp	olete
See also	UI32TO	32 RaH, 32 RaH, F32 Ral	, RbH , mem32				- •	



# F32TOUI16 RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer

Operands							
	RaH	floating-poin	t destination	register (R0H	to R7H)		
	RbH	floating-poin	nt source regi	ster (R0H to R	27H)		
Opcode	LSW: 1110 0110 MSW: 0000 0000						
Description	Convert the 32 truncate to zero nearest even v RaH(15:0) = F RaH(31:16) = 0	D. The result value use the 32ToUI16(RbH	will be stor F32TOUI1	ed in RaH.	To instead		
Flags	This instruction	does not affe	ect any fla	gs:			
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No	No	No	No	No	No	No
Pipeline	This is a 2 pipe F32TOUI16 NOP NOP Any instruction as a source op	RaH, RbH	; 2 p; ; 1 c; ; <	ipeline cyc ycle delay F32TOUI16	or non-conf completes,	RaH updated	1
Example	MOVIZF32 F32TOUI16 MOVIZF32 F32TOUI16 NOP	R6H, #-9.0 R7H, R6H	<pre>; R5H (15 ; R5H (31 ; R6H = -9 ; &lt; F32' ; ; R7H (15 ; R7H (31 ; 1 Cycle</pre>	:0) = F32TO :16) = 0x00 9.0 (0xC110 TOUI16 comp :0) = F32TO :16) = 0x00 delay for	UI16 (R4H) 00 0000) lete, R5H ( R5H ( UI16 (R6H) 00 F32TOUI16 t lete, R7H (	31:16) = 0. o complete	0 (0x0000) 0 (0x0000)
See also	F32TOI16 RaH F32TOUI16R F F32TOUI16R F I16TOF32 RaH I16TOF32 RaH UI16TOF32 Ra UI16TOF32 Ra	RaH, RbH RaH, RbH I, RbH I, mem16 IH, mem16					



# F32TOUI16R RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer and Round

Operands							
	RaH	floating-point d	estination r	egister (R0H	to R7H)		
	RbH	floating-point s	ource regis	ster (R0H to R	87H)		
Opcode	LSW: 1110 0110 MSW: 1000 0000	1000 1110 00bb baaa					
Description	Convert the 32- the closest even converted value RaH(15:0) = F3 RaH(31:16) = 0x	value. The re , use the F32T	sult will t OUI16 ir	be stored ir	-	-	
Flags	This instruction						
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No	No	No	No	No	No	No
Pipeline	This is a 2 pipel F32TOUI16R NOP NOP		; 2 p ; 1 c	ipeline cy ycle delay	or non-cor	nflicting ir s, RaH updat	
	Any instruction i as a source ope		ot must no	ot use Ra⊦	l as a destir	nation regist	er or use RaH
Example	MOVIZ MOVXI	R5H, #0x412C R5H, #0xCCCD	; R5H	= 0x412C = 0xCCCD = 10.8 (0x	412CCCCD)		
	F32TOUI16R	R6H, R5H	; R6H		F32TOUI16rc	ound (R5H)	
	MOVF32	R7H, #-10.8	; R7H ; < ; R6H	= -10.8 (0 F32TOUI16R (15:0) =	x0xCl2CCCCI complete, 11.0 (0x000 0.0 (0x000	)B)	
	F32TOUI16R	ROH, R7H	; R0H		F32TOUI16rc		
	NOP		; 1 Cy ; < ; ROH	cle delay F32TOUI16R (15:0) =			plete
See also	F32TOI16 RaH, F32TOI16R RaH F32TOUI16 RaH I16TOF32 RaH, I16TOF32 RaH, UI16TOF32 RaH UI16TOF32 RaH	H, RbH H, RbH RbH mem16 H, mem16					



# F32TOUI32 RaH, RbH Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer

Operands									
	RaH		floating-poir	nt destinatior	register (R0H	to R7H)			
	RbH		floating-poir	nt source reg	ister (R0H to F	R7H)			
Opcode	LSW: 111 MSW: 000		1000 1010 00bb baaa						
Description	Convert result in RaH = 1	RaH.		oint value	in RbH to a	n unsigned	32-bit intege	er and store the	
Flags	This ins	truction	does not aff	ect any fla	gs:				
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	•
	Modified	No	No	No	No	No	No	No	_
Pipeline	F32 NOP NOP	FOUI32	ine cycle (2р <sup>кан, кbн</sup> in the delav	; 2 pi ; 1 cy ; <	peline cycl cle delay c F32TOUI32 c	or non-confl completes, F	-	ruction er or use RaH	
	as a sou								
Example	F32 MOV	IZF32 TOUI32 IZF32 TOUI32	R6H, #12.5 R7H, R6H R1H, #-6.5 R2H, R1H	; R7H = ; R1H = ; < F3 ; R2H = ; 1 Cycl	F32TOUI32 ( -6.5 (0xC0I 2TOUI32 com F32TOUI32 ( e delay for	R6H) 000000) plete, R7H R1H) F32TOUI32	= 12.0 (0x0 to complete = 0.0 (0x00	1	
See also	UI32TO	32 RaH 32 RaH F32 Ral	RbH mem32						



# FRACF32 RaH, RbH Fractional Portion of a 32-bit Floating-Point Value

Operanus									
	RaH		floating-point	destination	register (R0H	to R7H)			
	RbH		floating-point	source regi	ster (R0H to R	7H)			
Opcode	LSW: 111 MSW: 000		1111 0001 00bb baaa						
Description	Returns	in RaH t	the fractional	portion c	f the 32-bit	floating-poi	nt value in F	RpH	
Flags	This inst	ruction o	does not affe	ct any fla	gs:				
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	No	No	
Pipeline		<b>2 pipeli</b> F32 Rai	ne cycle (2p) н, кbн	; 2 pig ; 1 cyc	peline cycl cle delay o		icting inst updated	ruction	
	Any insti as a sou			lot must r	ot use RaH	l as a destir	nation regist	er or use Ral	-1
Example	MOVI FRAC NOP		2H, #19.625 3H, R2H	; R3H = ; 1 Cyc	-	2H) r FRACF32 t	o complete 0.625 (0x3	F200000)	
See also									



# I16TOF32 RaH, RbH Convert 16-bit Integer to 32-bit Floating-Point Value

Operands								
	RaH	floating-poir	nt destination	n register (R0H	I to R7H)			
	RbH	floating-poir	nt source reg	ister (R0H to F	R7H)			
Opcode	LSW: 1110 ( MSW: 0000 (							
Description	result in Ra		teger in R	bH to a 32-l	bit floating p	oint value a	nd store the	
	RaH = I16To	F32 RbH						
Flags	This instruc	tion does not aff	ect any fla	igs:				
	Flag TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified No	o No	No	No	No	No	No	
Pipeline	I16TOF3 NOP NOP	bipeline cycle (2) B2 RaH, RbH tion in the delay e operand.	; 2 pi ; 1 cy ; <	peline cycl cle delay o I16TOF32 co	les (2p) or non-confi ompletes, Ra	-		I
Example	MOVIZ MOVXI	R0H, #0x0000 R0H, #0x0004 R2 R1H, R0H R2H, #0x0000 R2H, #0xFFFC R2H, R2H	<pre>H ; R0H[1 ; R1H = ) ; R2H[3 ; <i1 1="" ;="" c="" cyc<="" pre="" r2h[1="" r3h=";"></i1></pre>	5:0] = -4.0 I16TOF32 ( le delay fo	0 (0x0004) (R0H) 0 (0x0000) plete, R1H 0 (0xFFFC) (R2H) 0r I16TOF32	= 4.0 (0x408 to complete = -4.0 (0x0	1	
See also	116TOF32 F	RaH, RbH RaH, RbH R RaH, RbH RaH, mem16 RaH, mem16						



# I16TOF32 RaH, mem16 Convert 16-bit Integer to 32-bit Floating-Point Value

Operands							
	RaH	floating-point	t destinatior	n register (R0H	I to R7H)		
	mem316	16-bit source	e memory lo	ocation to be co	onverted		
Opcode	LSW: 1110 01 MSW: 0000 0a						
Description		16-bit signed into t value and store			e mem16 po	inter to a 32	-bit
Flags	This instructi	on does not affe	ect any fla	igs:			
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No	No	No	No	No	No	No
Pipeline	I16TOF32 NOP NOP	ipeline cycle (2p RaH, mem16 on in the delay s operand.	; 2 pi ; 1 cy ; <	peline cycl cle delay o I16TOF32 co	les (2p) or non-confi ompletes, Ra		
Example	MOVW MOV I16TOF32 MOV I16TOF32 NOP	@1, #0xFFFC	; [0x00 ; R0H = ; [0x00 ; <i1 ; R1H = ; 1 Cyc</i1 	A000] = 4.0 I16TOF32   A001] = -4. 6TOF32 comp I16TOF32   le delay fo	[0x00A000] .0 (0xFFFC) plete, R0H [0x00A001] pr I16TOF32	= 4.0 (0x408 to complete = -4.0 (0x0	2
See also	F32TOI16 R F32TOI16R F32TOUI16 F32TOUI16F I16TOF32 R UI16TOF32 UI16TOF32	RaH, RbH RaH, RbH R RaH, RbH aH, RbH RaH, mem16					

# I32TOF32 RaH, mem32 Convert 32-bit Integer to 32-bit Floating-Point Value

Operands											
	RaH		floating-poin	t destinatior	register (R0H	to R7H)					
	mem32	mem32 32-bit source for the MOV32 operation. mem32 means that the operation can only address memory using any of the direct or indirect addressing modes supported by the C28x CPU									
Opcode	LSW: 1110 MSW: 0000		1000 1000 mem32								
Description		ue and	bit signed int store the res			mem32 po	inter to a 32-	-bit floating			
Flags	This instr	uction	does not affe	ect any fla	gs:						
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No	No	No			
	•			; <	I32TOF32 cc	mpletes, Ra	-	er or use RaH			
Example	AS A SOUR MOVW MOV MOV I32TC NOP	L (G (G	PP, #0x0280 00, #0x1111 01, #0x1111 R1H, @0	<pre>; [0x00A ; [0x00A ; Value ; 0x00A0 ; R1H = ; 1 Cycl</pre>	000] = 4369 001] = 4369 of the 32 b 01 and 0x00 I32TOF32 (0 e delay for	(0x1111) oit signed : A000 is +28 x11111111) I32TOF32 1	integer pres 36331153 (02 to complete = 286331153				
See also	F32T013 F32T0U1 I32T0F3 U132T0F U132T0F	32 Rai 2 RaH, 32 Rai	H, RbH RbH	15		,					



# I32TOF32 RaH, RbH Convert 32-bit Integer to 32-bit Floating-Point Value

Operands								
	RaH		floating-point	destination	register (R0H	to R7H)		
	RbH		floating-point	source reg	ister (R0H to F	87H)		
Opcode	LSW: 111 MSW: 000		1000 1001 00bb baaa					
Description	Convert result in RaH = 13	RaH.	ned 32-bit inte	eger in R	bH to a 32-t	bit floating-p	oint value ai	nd store the
Flags	This inst	truction	does not affe	ect any fla	igs:			
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
Pipeline	I327 NOP NOP	TOF32 R	ine cycle (2р ан, Rbн	; 2 pi ; 1 cy ; <	peline cycl cle delay c I32TOF32 cc	r non-confl mpletes, Ra	-	
	Any inst as a sou			lot must	not use RaF	l as a destir	nation registe	er or use RaH
Example	MOVI MOVX I327 NOP	KI R	2H, #0x1111 2H, #0x1111 3H, R2H	; R2H[1 ; Value ; in R2 ; R3H = ; 1 Cyc	H is +28633 I32TOF32 ( le delay fo	9 (0x1111) bit signed 1153 (0x111 R2H) r I32TOF32	to complete	
See also	F32T013 F32T0U 132T0F3 U132T01 U132T01	JI32 Rai 32 RaH, F32 Rai	H, RbH mem32	, < 1	5210F52 COI	piece, KSU	- 200331133	(074000000)

# MACF32 R3H, R2H, RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Add

Operands					Ilel multiply			e operands are	
			, RaH, RbH , R3H, R2H						
	R3H		floating-poir	nt destination	and source reg	gister for the A	DDF32		
	R2H		floating-poir	nt source reg	ister for the AD	DF32 operatio	on (R0H to R7H	I)	
	RdH		floating-poir RdH cannot		register for MF	YF32 operati	on (R0H to R7H	H)	
	ReH		floating-poir	nt source reg	ister for MPYF3	2 operation (I	R0H to R7H)		
	RfH		floating-poir	nt source reg	ister for MPYF3	2 operation (I	R0H to R7H)		
Opcode	LSW: 111 MSW: fee		0100 00ff ccbb baaa						
Description	This inst instruction		s an alias fo	or the para	llel multiply	and add, M	IACF32    AE	DDF32,	
	RdH = Re R3H = R3								
Restrictions	The des cannot b		egister for	the MPYF	32 and the A	.DDF32 mu	ist be unique	e. That is, RdH	
Flags	This inst	ruction r	nodifies the	following	flags in the	STF registe	er:.		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	Yes	Yes	
	The STF	- register	r flags are r	nodified as	s follows:				
	<ul> <li>LUF = 1 if MPYF32 or ADDF32 generates an underflow condition.</li> </ul>								
					nerates an o				
Pipeline	Both MF	YF32 ar	nd ADDF32	take 2 pir	eline cycles	(2p) That i	s:		
			, RbH, RcH		eline cycles		•••		
	ADDE NOP	32 RdH	, ReH, RfH		eline cycles		ating instr	wation	
	NOP				le delay or PYF32, ADDF3				
	NOP								
	Any inst a source			slot must i	not use RaH	or RdH as	a destinatio	n register or as	



Example	; Perform 5 multiply and accumulate operations:
	; 1st multiply: A = X0 * Y0
	; 2nd multiply: B = X1 * Y1
	; 3rd multiply: C = X2 * Y2
	; 4th multiply: D = X3 * Y3
	; 5th multiply: E = X3 * Y3
	; ; Result = A + B + C + D + E
	MOV32 R0H, *XAR4++ ; R0H = X0
	MOV32 R1H, $*XAR5++$ ; R1H = Y0
	; $R2H = A = X0 * Y0$
	MPYF32 R2H, R0H, R1H ; In parallel R0H = X1
	MOV32 R0H, *XAR4++
	MOV32 R1H, *XAR5++ ; R1H = Y1
	; R3H = B = X1 * Y1
	MPYF32 R3H, R0H, R1H ; In parallel R0H = X2
	MOV32 ROH, *XAR4++
	MOV32 R1H, *XAR5++ ; R1H = Y2
	; $R3H = A + B$ ; $R2H = C = X2 * Y2$
	MACF32 R3H, R2H, R2H, R0H, R1H ; In parallel R0H = X3
	MOV32 ROH, *XAR4++
	MOV32 R1H, *XAR5++ ; R1H = Y3
	; $R3H = (A + B) + C$
	; $R2H = D = X3 * Y3$
	MACF32 R3H, R2H, R2H, R0H, R1H ; In parallel R0H = X4
	MOV32 R0H, *XAR4
	MOV32 R1H, *XAR5 ; R1H = Y4
	; The next MACF32 is an alias for ; MPYF32    ADDF32
	; $R2H = E = X4 * Y4$
	MACF32 R3H, R2H, R2H, R0H, R1H ; in parallel R3H = (A + B + C) + D NOP ; Wait for MPYF32    ADDF32 to complete
	ADDF32 R3H, R3H, R2H ; R3H = $(A + B + C + D) + E$
	NOP ; Wait for ADDF32 to complete MOV32 @Result, R3H ; Store the result
	MOV32 @Result, RSH , Store the result
See also	MACF32 R3H, R2H, RdH, ReH, RfH    MOV32 RaH, mem32
	MACF32 R7H, R3H, mem32, *XAR7++
	MACF32 R7H, R6H, RdH, ReH, RfH
	MACF32 R7H, R6H, RdH, ReH, RfH    MOV32 RaH, mem32
	MPYF32 RaH, RbH, RcH    ADDF32 RdH, ReH, RfH



### MACF32 R3H, R2H, RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Multiply and Accumulate with Parallel Move

Operands									
	R3H	flo	ating-point	destination/s	source registe	r R3H for the a	add operation		
	R2H	R2Hfloating-point source register R2H for the add operationRdHfloating-point destination register (R0H to R7H) for the multiply operation RdH cannot be the same register as RaH							
	RdH								
	ReH	H floating-point source register (R0H to R7H) for the multiply operation							
	RfH	floating-point source register (R0H to R7H) for the multiply operation							
	RaH				register for the e same regist		ation (R0H to R	7H).	
	mem32	32	2-bit source	for the MOV	/32 operation				
Opcode	LSW: 1110 MSW: eedd		.1 fffe nem32						
Description		. The des	tination re	egister for			s and move the same a	from register s the	
	R3H = R3H RdH = ReH RaH = [mem	* RfH,							
Restrictions						MOV32 mu ster as RdH		e. That is, RaH	
Flags	This instru	ction mod	ifies the f	following f	lags in the	STF registe	r:		
	Flag T	F Z	(1	NI	ZF	NF	LUF	LVF	
	Modified N	lo Y	′es	Yes	Yes	Yes	Yes	Yes	
	• LVF = '	1 if MACF 1 if MACF ts the NF, 1); 23) == 0) 1);	32 (add c 32 (add c , ZF, NI a { zF =	or multiply or multiply and ZI flag	r) generates ) generates s as follows	an overflow	ow conditior v condition.	l.	
Pipeline	The MACF	32 takes	2 pipeline	e cycles (2	2p) and the	MOV32 tak	es a single	cycle. That is:	
	MACF32    MOV32 NOP	R3H, R2 RaH, me		ReH, RfH	; 1 cycl ; < MO ; 1 cycl	V32 complet e delay for	es, RaH upd		
	NOP								
					version of s a source		ust not use	R3H or RdH as	



Example	; Perform 5 multiply and accumulate operations:
	; ; 1ST multiply: A = X0 * Y0 ; 2nd multiply: B = X1 * Y1 ; 3rd multiply: C = X2 * Y2 ; 4TH multiply: D = X3 * Y3 ; 5th multiply: E = X3 * Y3
	; ; Result = A + B + C + D + E
	MOV32 R0H, *XAR4++ ; R0H = X0 MOV32 R1H, *XAR5++ ; R1H = Y0
	<pre></pre>
	; R3H = B = X1 * Y1 MPYF32 R3H, R0H, R1H ; In parallel R0H = X2    MOV32 R0H, *XAR4++
	MOV32 R1H, *XAR5++ ; R1H = Y2
	; R3H = A + B ; R2H = C = X2 * Y2 MACF32 R3H, R2H, R2H, R0H, R1H ; In parallel R0H = X3    MOV32 R0H, *XAR4++
	MOV32 R1H, *XAR5++ ; R1H = Y3 ; R3H = (A + B) + C
	; R2H = D = X3 * Y3 MACF32 R3H, R2H, R2H, R0H, R1H ; In parallel R0H = X4    MOV32 R0H, *XAR4 MOV32 R1H, *XAR5 ; R1H = Y4
	; R2H = E = X4 * Y4
	MPYF32 R2H, R0H, R1H ; in parallel R3H = (A + B + C) + D    ADDF32 R3H, R3H, R2H
	NOP ; Wait for MPYF32    ADDF32 to complete
	ADDF32 R3H, R3H, R2H; R3H = (A + B + C + D) + ENOP; Wait for ADDF32 to completeMOV32 @Result, R3H; Store the result
See also	MACF32 R3H, R2H, RdH, ReH, RfH MACF32 R7H, R3H, mem32, *XAR7++ MACF32 R7H, R6H, RdH, ReH, RfH MACF32 R7H, R6H, RdH, ReH, RfH    MOV32 RaH, mem32 MPYF32 RaH, RbH, RcH    ADDF32 RdH, ReH, RfH



# MACF32 R7H, R3H, mem32, \*XAR7++ 32-bit Floating-Point Multiply and Accumulate

### Operands

Operands											
	R7H		floating-p	oint destinatior	register						
	R3H		floating-p	oint destinatior	register						
	mem32	mem32 pointer to a 32-bit source location									
	*XAR7										
Opcode	LSW: 111 MSW: 00b		0101 000 mem32	0							
Description				ccumulate c a single mul			s a stand-ald	one operation,			
	Cycle 1:	R3H =	R3H + R2H	[, R2H = [me	m32] * [XAR	7++]					
	repeat in alternate tempora <sub>Cycle 1</sub> :	This instruction is the only floating-point instruction that can be repeated using the single repeat instruction (RPT   ). When repeated, the destination of the accumulate will alternate between R3H and R7H on each cycle and R2H and R6H are used as temporary storage for each multiply. Cycle 1: R3H = R3H + R2H, R2H = [mem32] * [XAR7++] Cycle 2: R7H = R7H + R6H, R6H = [mem32] * [XAR7++]									
	Cycle 4: etc	R7H =	R7H + R6H	I, R2H = [me I, R6H = [me	m32] * [XAR	.7++]					
Restrictions	R2H and R6H will be used as temporary storage by this instruction.										
Flags	This inst	ruction	modifies t	he following	flags in the	STF registe	r:				
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No	Yes	Yes			
Pipeline	<ul> <li>LUF</li> <li>LVF :</li> <li>When reinstruction restriction </li> <li>inst </li> <li>inst </li> <li>MACF3</li> </ul>	= 1 if M = 1 if M epeated on is repond function function (N-1) 22 R7H,	ACF32 ge ACF32 ge the MACF beated. W	e modified as enerates an o 532 takes 3 - hen repeate	underflow co overflow cor + N cycles v d, this instru ; No re ; Canno ; to R2 ; Execu	ndition. where N is the action has the striction t be a 2p i H, R3H, R6F te N times,	e following p nstruction or R7H where N is	bipeline			
	<inst< td=""><td>ructior</td><td>13&gt;</td><td></td><td></td><td>strictions.</td><td>U P6U and</td><td>ס7ט</td></inst<>	ructior	13>			strictions.	U P6U and	ס7ט			

; Can read R2H, R3H, R6H and R7H



Example

MACF32 can also be used standalone. In this case, the insruction takes 2 cycles and the following pipeline restrictions apply:

<instruction1> <instruction2> MACF32 R7H, R3H, *XAR6, *XAR7</instruction2></instruction1>	<pre>; No restriction ; Cannot be a 2p instruction that writes ; to R2H, R3H, R6H or R7H ; R3H = R3H + R2H, R2H = [mem32] * [XAR7++] ; &lt; R2H and R3H are valid (note: no</pre>
delay required)	
NOP	
ZERO R2H ZERO R3H ZERO R6H ZERO R7H	; Zero the accumulation registers ; and temporary multiply storage registers
RPT #3	; Repeat MACF32 N+1 (4) times
MACF32 R7H, R3H, *XAR6++, *XAR7++	-
ADDF32 R7H, R7H, R3H	; Final accumulate
NOP	; < ADDF32 completes, R7H valid

Cascading of RPT || MACF32 is allowed as long as the first and subsequent counts are even. Cascading is useful for creating interruptible windows so that interrupts are not delayed too long by the RPT instruction. For example:

	ZERO R2H ZERO R3H		Zero the accumulation registers and temporary multiply storage registers
	ZERO R6H ZERO R7H		
	RPT #3	;	Execute MACF32 N+1 (4) times
	MACF32 R7H, R3H, *XAR6++, *XAR7++ RPT #5	;	Execute MACF32 N+1 (6) times
	MACF32 R7H, R3H, *XAR6++, *XAR7++		
11	RPT #N MACF32 R7H, R3H, *XAR6++, *XAR7++	;	Repeat MACF32 N+1 times where N+1 is even
11	ADDF32 R7H, R7H, R3H	;	Final accumulate
	101	;	< ADDF32 completes, R7H valid

See also

MACF32 R3H, R2H, RdH, ReH, RfH || MOV32 RaH, mem32 MACF32 R7H, R6H, RdH, ReH, RfH || MOV32 RaH, mem32 MPYF32 RaH, RbH, RcH || ADDF32 RdH, ReH, RfH



# MACF32 R7H, R6H, RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Add

Operands	This instruction is an alias for the parallel multiply and add instruction. The operands are translated by the assembler such that the instruction becomes:									
			, RaH, RbH , R7H, R6H							
	R7H floating-point destination and source register for the ADDF32									
	R6H floating-point source register for the ADDF32 operation (R0H to R7H)									
	RdH		floating-poir RdH cannot		register for MP	YF32 operation	on (R0H to R7H	1)		
	ReH		floating-poir	nt source reg	ister for MPYF3	2 operation (F	R0H to R7H)			
	RfH		floating-poir	nt source reg	ister for MPYF3	2 operation (F	R0H to R7H)			
Opcode	LSW: 111 MSW: fee		0100 00ff ccbb baaa							
Description	This inst instruction		s an alias fo	or the para	llel multiply a	and add, M	ACF32    AE	)DF32,		
	RdH = Ra R7H = R6									
Restrictions	The des cannot b		egister for	the MPYF	32 and the A	DDF32 mu	ist be unique	e. That is, RdH		
Flags	This inst	ruction r	nodifies the	following	flags in the S	STF registe	er:.			
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No	Yes	Yes		
	<b>T</b> 0 <b>T</b>	<u> </u>	4		<b>6</b> H					
		-	flags are r							
				0	nerates an u					
	• LVF	= 1 if MF	PYF32 or A	DDF32 ge	nerates an o	verflow cor	ndition.			
Pipeline	Both MF	YF32 ar	nd ADDF32	take 2 pip	eline cycles	(2p) That i	s:			
			, RbH, RcH		eline cycles	-				
	ADDF NOP	32 RdH	, ReH, RfH		eline cycles le delay or		cting instr	ruction		
				-	PYF32, ADDF3					
	NOP									
	Any inst a source			slot must i	not use RaH	or RdH as	a destinatio	n register or as		



Example	; Perform 5 multiply and accumulate operations:
	<pre>; 1st multiply: A = X0 * Y0 ; 2nd multiply: B = X1 * Y1 ; 3rd multiply: C = X2 * Y2 ; 4th multiply: D = X3 * Y3 ; 5th multiply: E = X3 * Y3</pre>
	; ; Result = A + B + C + D + E
	MOV32 R0H, *XAR4++ ; R0H = X0 MOV32 R1H, *XAR5++ ; R1H = Y0 ; R6H = A = X0 * Y0
	MPYF32 R6H, R0H, R1H ; In parallel R0H = X1    MOV32 R0H, *XAR4++ MOV32 R1H, *XAR5++ ; R1H = Y1
	<pre>; R7H = B = X1 * Y1 ; In parallel R0H = X2    MOV32 R0H, *XAR4++ MOV32 R1H, *XAR5++ ; R1H = Y2 ; R7H = A + B</pre>
	; R6H = C = X2 * Y2 MACF32 R7H, R6H, R6H, R0H, R1H ; In parallel R0H = X3    MOV32 R0H, *XAR4++ MOV32 R1H, *XAR5++ ; R1H = Y3 ; R7H = (A + B) + C
	; R6H = D = X3 * Y3 MACF32 R7H, R6H, R6H, R0H, R1H ; In parallel R0H = X4    MOV32 R0H, *XAR4 MOV32 R1H, *XAR5 ; R1H = Y4
	; Next MACF32 is an alias for ; MPYF32    ADDF32
	MACF32 R7H, R6H, R6H, R0H, R1H ; R6H = E = X4 * Y4 ; in parallel R7H = (A + B + C) + D ; Wait for MPYF32    ADDF32 to complete ADDF32 R7H, R7H, R6H ; R7H = (A + B + C + D) + E NOP ; Wait for ADDF32 to complete MOV32 @Result, R7H ; Store the result
See also	MACF32 R3H, R2H, RdH, ReH, RfH MACF32 R3H, R2H, RdH, ReH, RfH    MOV32 RaH, mem32 MACF32 R7H, R3H, mem32, *XAR7++ MACF32 R7H, R6H, RdH, ReH, RfH    MOV32 RaH, mem32 MPYF32 RaH, RbH, RcH    ADDF32 RdH, ReH, RfH



# MACF32 R7H, R6H, RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Multiply and Accumulate with Parallel Move

•								
Operands	. <u></u>							
	R7H	floating-po	pint destination	source registe	r R7H for the	add operation		
	R6H	floating-po	pint source regi	ster R6H for th	e add operati	on		
	RdH		pint destination not be the same			e multiply opera	ation.	
	ReH	floating-po	pint source regi	ster (R0H to R	7H) for the m	ultiply operation	l	
	RfH	floating-po	pint source regi	ster (R0H to R	7H) for the m	ultiply operation	I	
	RaH		oint destination ot be R3H or t			ation (R0H to F	R7H).	
	mem32	32-bit sou	Irce for the MO	V32 operation				
Opcode Description	LSW: 1110 00 MSW: eedd da Multiply/accu memory. The registers for f R7H = R7H + RdH = ReH * RaH = [mem32	aa mem32 mulate the co destination r he MACF32. R6H RfH,	ontents of flo					
Restrictions			or the MACF				e. That is, F	RaH
	cannot be R7	'H and RaH c	annot be the	e same regi	ster as RdH	۱	e. That is, F	RaH
Restrictions Flags	cannot be R7		annot be the	e same regi	ster as RdH	۱	e. That is, F	RaH
	cannot be R7	'H and RaH c	annot be the	e same regi	ster as RdH	۱	e. That is, F	RaH
	cannot be R7	'H and RaH o on modifies th	annot be the	e same regis flags in the s	ster as RdH STF registe	H. er:		{аН
	cannot be R7 This instruction Flag TF Modified No The STF regination • LUF = 1 if • LVF = 1 if The MOV32 NF = RaH(31) ZF = 0; if(RaH(30:23) NI = RaH(31) ZI = 0;	TH and RaH c on modifies th Yes ster flags are f MACF32 (ac MACF32 (ac Instruction wil ; ) == 0) {ZF =	annot be the ne following <u>NI</u> Yes modified as d or multiply d or multiply ll set the NF = 1; NF = 0	e same regis flags in the s Yes follows: y) generates y) generates , ZF, NI and	ster as RdH STF registe <u>NF</u> Yes s an underf s an overflo	I. Er: Ves Now condition w condition.	LVF Yes	{аН
	cannot be R7 This instruction Flag TF Modified No The STF regin • LUF = 1 if • LVF = 1 if The MOV32 NF = RaH(31) ZF = 0; if (RaH(30:23) NI = RaH(31) ZI = 0; if (RaH(31:0))	TH and RaH c on modifies th <u>ZI</u> Yes ster flags are f MACF32 (ac MACF32 (ac Instruction wil ; ) == 0) {ZF =	me following NI Yes modified as d or multiply d or multiply ll set the NF = 1; NF = 0 1;	e same regis flags in the s Yes follows: y) generates y) generates y, ZF, NI and	ster as RdH STF registe <u>NF</u> Yes an underf an overflo	I. LUF Yes low condition w condition. follows:	LVF Yes	



Example	; Perform 5 multiply and accumulate operations:
	<pre> / Ist multiply: A = X0 * Y0 / 2nd multiply: B = X1 * Y1 / 3rd multiply: C = X2 * Y2 / 4th multiply: D = X3 * Y3 / 5th multiply: E = X3 * Y3 /</pre>
	; Result = $A + B + C + D + E$
	MOV32 R0H, *XAR4++ ; R0H = X0 MOV32 R1H, *XAR5++ ; R1H = Y0
	; R6H = A = X0 * Y0
	MPYF32 R6H, R0H, R1H ; In parallel R0H = X1
	MOV32 R1H, *XAR5++ ; R1H = Y1
	; R7H = B = X1 * Y1
	MPYF32 R7H, R0H, R1H ; In parallel R0H = X2
	MOV32 R0H, *XAR4++ MOV32 R1H, *XAR5++ ; R1H = Y2
	; R7H = A + B ; R6H = C = X2 * Y2 MACF32 R7H, R6H, R6H, R0H, R1H ; In parallel R0H = X3    MOV32 R0H, *XAR4++ MOV32 R1H, *XAR5++ ; R1H = Y3
	; R7H = (A + B) + C ; R6H = D = X3 * Y3
	MACF32 R7H, R6H, R6H, R0H, R1H ; In parallel R0H = X4
	MOV32 R1H, *XAR5 ; R1H = Y4
	; R6H = E = X4 * Y4 MPYF32 R6H, R0H, R1H ; in parallel R7H = (A + B + C) + D    ADDF32 R7H, R7H, R6H
	NOP ; Wait for MPYF32    ADDF32 to complete
	ADDF32 R7H, R7H, R6H ; R7H = (A + B + C + D) + E NOP ; Wait for ADDF32 to complete
	MOV32 @Result, R7H ; Store the result
See also	MACF32 R7H, R3H, mem32, *XAR7++ MACF32 R3H, R2H, RdH, ReH, RfH    MOV32 RaH, mem32 MPYF32 RaH, RbH, RcH    ADDF32 RdH, ReH, RfH



# MAXF32 RaH, RbH 32-bit Floating-Point Maximum

	RaH		floating-po	int source/de	stination registe	r (R0H to R7H	)		
	RbH		floating-po	int source rec	jister (R0H to R	7H)			
pcode	LSW: 111 MSW: 000								
Description	if(RaH <	if(RaH < RbH) RaH = RbH							
	Special cases for the output from the MAXF32 operation:								
		-	will be conv ized output		finity verted to pos	sitive zero.			
Flags	This instruction modifies the following flags in the STF register:								
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	Yes	Yes	No	No	
	in the de if(RaH =	stinati	on register. {ZF=1, NF=	0}	on the result	of the opera	ition, not the	e result stored	
	<pre>in the de if(RaH = if(RaH &gt;</pre>	stinati = RbH) RbH)	on register.	0} 0}	in the result	of the opera	ition, not the	e result stored	
ipeline	<pre>in the de if(RaH = if(RaH &gt; if(RaH &lt;</pre>	estinati = RbH) RbH)	on register. {ZF=1, NF= {ZF=0, NF=	0} 0} 1}	n the result	of the opera	ition, not the	e result stored	
Pipeline Example	<pre>in the de if(RaH = if(RaH &gt; if(RaH &lt;</pre>	estinati = RbH) > RbH) < RbH) a single xF32	on register. {ZF=1, NF= {ZF=0, NF= {ZF=0, NF=	0} 0} 1} iction. 0 ; R1H 5 ; R2H ; R2H ; R1H ; R2H	n the result = 5.0 (0x = -2.0 (0x = -1.5 (0x = -1.5, ZF = -1.5, ZF = 5.0, ZF = 5.0, ZF	40A00000) C0000000) BFC00000) = NF = 0 = 0, NF = = 0, NF =	1	e result stored	



# MAXF32 RaH, #16FHi 32-bit Floating-Point Maximum

#### Operands

Operands											
	RaH	floating-p	oint source/de	stination registe	er (R0H to R7H	I)					
	#16FHi				nts the upper 16 the mantissa a						
Opcode	LSW: 1110 10 MSW: IIII II										
Description	immediate va	Compare RaH with the floating-point value represented by the immediate operand. If the immediate value is larger, then load it into RaH. if(RaH < #16FHi:0) RaH = #16FHi:0									
	#16FHi is a 1 floating-point addressing m are 0. Some -1.5 (0xBFC0	#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x4000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.									
	Special cases	Special cases for the output from the MAXF32 operation:									
		ut will be cor alized output			sitive zero.						
Flags	This instruction modifies the following flags in the STF register:										
	Flag TF	ZI	NI	ZF	NF	LUF	LVF				
	Modified No	No	No	Yes	Yes	No	No				
	The ZF and N in the destina if(RaH == #1 if(RaH > #1 if(RaH < #1	tion register. 5FHi:0) {ZF= 5FHi:0) {ZF=	1, NF=0} 0, NF=0}	on the result	of the opera	ation, not the	e result stored				
Pipeline	This is a sing	le-cycle instr	uction.								
Example	MOVIZF32 MOVIZF32 MOVIZF32 MAXF32 MAXF32	ROH, #5. R1H, #4. R2H, #-1 ROH, #5. R1H, #2.	0 ; R1H .5 ; R2H 5 ; R0H	I = 5.0 (0) $I = 4.0 (0)$ $I = -1.5 (0)$ $I = 5.5, ZH$ $I = 4.0, ZH$	(40800000) (BFC00000) F = 0, NF =						
	MAXF32 MAXF32 MAXF32	R2H, #-1 R2H, #-1			F = 0, NF = F = 1, NF =	1					

Submit Documentation Feedback



Instructions

www.ti.com

# MAXF32 RaH, RbH ||MOV32 RcH, RdH 32-bit Floating-Point Maximum with Parallel Move

Operands									
	RaH			point source/des			(F32 operation (	R0H to R7H)	
	RbH		floating-p	oint source regi	ster for the	MAXF32 operat	ion (R0H to R7H	H)	
	RcH			point destination			eration (R0H to I	R7H)	
	RdH		floating-p	point source regi	ster for the	MOV32 operation	on (R0H to R7H	)	
Opcode	LSW: 111 MSW: 000		1001 110 ccbb baa						
Description		n value	. If RaH is	hen load Ral less than R					
	if(RaH <	RbH)	{ RaH = Rk	oH; RcH = Rdl	H; }				
	because	of the	IEEE float	s performed ing-point forr the floating-	nat offset	s the expone	•	•	
	Special of	cases fo	or the outp	out from the I	MAXF32	operation:			
		•		nverted to inf t will be conv	•	ositive zero.			
Restrictions	The dest	ination	rogistor fo	r the MAVE		MOV/22 m	at ha uniqua	That is D	
Restrictions				or the MAXF: ter as RcH.	32 and the		ist be unique	. That is, R	ап
	cannot b	e the s	ame regis	ter as RcH.				. mai 15, Ki	ап
Flags	cannot b	e the s ruction	ame regis modifies t	ter as RcH. he following	flags in th	ie STF regist	er:		an 
	cannot b	e the s	ame regis	ter as RcH.				LVF	an 
Flags	cannot b This insti Flag Modified The ZF a in the de if (RaH = if (RaH >	e the s ruction TF No and NF stinatio = RbH) RbH) RbH)	ame regis modifies t ZI No flags are on register. {ZF=1, NF {ZF=0, NF {ZF=0, NF	ter as RcH. he following NI No configured of r=0 } r=1 }	flags in th <b>ZF</b> Yes	ne STF regist NF Yes	er: LUF No	LVF No	
	cannot b This insti Flag Modified The ZF a in the de if (RaH = if (RaH >	e the s ruction TF No and NF stinatio = RbH) RbH) RbH)	ame regis modifies t ZI No flags are flags are son register. {ZF=1, NF {ZF=0, NF	ter as RcH. he following NI No configured of r=0 } r=1 }	flags in th <b>ZF</b> Yes	ne STF regist NF Yes	er: LUF No	LVF No	
Flags	cannot b This insti Flag Modified The ZF a in the de if (RaH = if (RaH >	e the s ruction TF No and NF stinatic = RbH) RbH) Single- ZF32 ZF32 ZF32 ZF32 ZF32 ZF32 32 2 32 32	ame regis modifies t ZI No flags are on register. {ZF=1, NF {ZF=0, NF {ZF=0, NF	ter as RcH. he following NI No configured of r=0 } r=0 } r=1 } ruction. 5.0 ; R01 1.0 ; R11 1.1.5 ; R21 2.0 ; R31 H ; R01 2H 2H ; R01 2H 2H ; R01	flags in the ZF Yes The result of the resul	ne STF regist NF Yes	er: <u>LUF</u> No ration, not the ZF = 0, NF = ZF = 0, NF	LVF No e result stor	



perands							
	RaH	floating-poin	t source/desti	nation registe	r (R0H to R7H)		
	RbH	floating-poin	t source regis	ter (R0H to R	.7H)		
ocode	LSW: 1110 0110 MSW: 0000 0000						
scription	if(RaH > RbH)	RaH = RbH					
	Special cases	for the output	from the M	INF32 ope	eration:		
	<ul> <li>NaN output</li> </ul>	will be conve	rted to infir	ity			
	<ul> <li>A denorma</li> </ul>	ized output w	ill be conve	erted to pos	sitive zero.		
Flags	This instructior	modifies the	following f	ags in the	STF register		
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No	No	No	Yes	Yes	No	No
	The ZF and NI		nfigured on	the result	of the opera	tion, not the	e result stored
	<pre>in the destinati if(RaH == RbH if(RaH &gt; RbH if(RaH &lt; RbH)</pre>	on register. {ZF=1, NF=0 {ZF=0, NF=0 {ZF=0, NF=1	} } }	the result	of the opera	tion, not the	e result stored
peline	<pre>in the destinati if(RaH == RbH) if(RaH &gt; RbH)</pre>	on register. {ZF=1, NF=0 {ZF=0, NF=0 {ZF=0, NF=1	} } }	the result	of the opera	tion, not the	e result store
ipeline xample	<pre>in the destinati if(RaH == RbH if(RaH &gt; RbH if(RaH &lt; RbH)</pre>	on register. {ZF=1, NF=0 {ZF=0, NF=0 {ZF=0, NF=1	<pre>} } tion. ; R0H ; R0H ; R1H ; R2H ; R0H ; R1H ; R2H ; R2H</pre>	= 5.0 (0x = 4.0 (0x = -1.5 (0x = 4.0, ZF = -1.5, ZF = -1.5, ZF	40A00000) 40800000)		e result stored

# MINF32 RaH, RbH 32-bit Floating-Point Minimum

# MINF32 RaH, #16FHi 32-bit Floating-Point Minimum

### Operands

Operands											
	RaH	floating-p	oint source/des	tination register	r (R0H to R7H	)					
	#16FHi			that represent low 16-bits of t							
Opcode	LSW: 1110 100 MSW: IIII III										
Description	immidate valu	Compare RaH with the floating-point value represented by the immediate operand. If the immidate value is smaller, then load it into RaH. if(RaH > #16FHi:0) RaH = #16FHi:0									
	floating-point addressing m are 0. Some e -1.5 (0xBFC0	#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.									
	Special cases for the output from the MINF32 operation:										
			verted to inf	inity rerted to pos	itive zero.						
Flags	This instruction modifies the following flags in the STF register:										
	Flag TF	ZI	NI	ZF	NF	LUF	LVF				
	Modified No	No	No	Yes	Yes	No	No				
	The ZF and N in the destina if(RaH == #16 if(RaH > #16 if(RaH < #16	tion register. FHi:0) {ZF= FHi:0) {ZF=	1, NF=0} 0, NF=0}	n the result o	of the opera	ation, not the	e result stored				
Pipeline	This is a singl	e-cycle instr	uction.								
Example	MOVIZF32 MOVIZF32 MOVIZF32 MINF32 MINF32 MINF32 MINF32	ROH, #5. R1H, #4. R2H, #-1 ROH, #5. R1H, #2. R2H, #-1 R2H, #-1	0 ; R1H .5 ; R2H 5 ; R0H 5 ; R1H .0 ; R2H	= 5.0 (0x4 = 4.0 (0x4 = -1.5 (0xH = 5.0, 2F = 2.5, 2F = -1.5, 2F = -1.5, 2F	40800000) 3FC00000) = 0, NF = = 0, NF = = 0, NF =	0 1					
See also	MAXF32 RaH MAXF32 RaH MINF32 RaH, MINF32 RaH,	, RbH RbH									

www.ti.com



### MINF32 RaH, RbH MOV32 RcH, RdH 32-bit Floating-Point Minimum with Parallel Move

Operands										
	RaH		floating-point s RaH cannot b			er for the MIN32 c RcH	peration (RC	)H to R7H)		
	RbH		floating-point :	source regist	er for the M	IN32 operation (R	0H to R7H)			
	RcH		floating-point of RcH cannot be			ne MOV32 operatio RaH	on (R0H to F	R7H)		
	RdH		floating-point :	source regist	er for the M	OV32 operation (F	R0H to R7H)			
Opcode	LSW: 111 MSW: 000		1001 1101 ccbb baaa							
Description	if(RaH >	RbH) {	RaH = RbH; H	RcH = RdH;	}					
	Special of	cases for	the output fi	om the M	NF32 op	eration:				
	NaN	output w	ill be convert	ed to infin	ty					
	A der	normalize	ed output will	be conve	rted to po	sitive zero.				
Restrictions			egister for the me register a		and the N	MOV32 must b	e unique.	That is, RaH		
Flags	This instruction modifies the following flags in the STF register:									
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified	No	No	No	Yes	Yes	No	No		
	<pre>in the de if(RaH = if(RaH &gt;</pre>	stination = RbH) { RbH) {	ags are conf register. ZF=1, NF=0} ZF=0, NF=0} ZF=0, NF=1}	igured on	the result	of the operation	on, not the	e result stored		
Pipeline	This is a	single-c	ycle instruction	on.						
Example	MOVI MOVI	2 32 2 32 32	R0H, #5.0 R1H, #4.0 R2H, #-1.5 R3H, #-2.0 R0H, R1H R3H, R2H R1H, R0H R3H, R2H R3H, R2H R2H, R1H R1H, R3H	; R1H ; R2H ; R3H ; R0H ; R1H	$= 4.0 (0)$ $= -1.5 (0)$ $= -2.0 (0)$ $= 4.0, R^{2}$ $= 4.0, R^{2}$	Dx40A00000) Dx40800000) DxBFC00000) DxC0000000) 3H = -1.5, ZF 3H = -1.5, ZF R1H = 4.0, ZF	= 1, NF =	- 0		
See also	MINF32 MINF32	RaH, Rb RaH, #1								



# MOV16 mem16, RaH Move 16-bit Floating-Point Register Contents to Memory

epolaliao									
	mem16	points to	the 16-bit dest	ination memory	,				
	RaH	floating-p	point source reg	jister (R0H to F	87H)				
Opcode	LSW: 1110 0 MSW: 0000 0		1						
Description		value from th nted to by me aH[15:0]		oits of the flo	ating-point	register (Ral	⊣[15:0]) to th	IE	
Flags	No flags STF flags are affected.								
	Flag TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified No	No	No	No	No	No	No		
Pipeline	This is a sir	gle-cycle insti	ruction.						
Example	MOVW MOVXI MOV16	DP, #0x02C0 R4H,#0x0003 @0, R4H	; R4H = 3.		x0003)				
See also	MOVIZF32	I, #16FHiHex RaH, #16FHi I, #16FLoHex							



# MOV32 \*(0:16bitAddr), loc32 Move the Contents of loc32 to Memory

Operatius								
	0:16bitAd	dr	16-bit immedi	ate addres	s, zero extende	ed		
	loc32		32 bit source	location				
Opcode	LSW: 101 MSW: III		loc32 IIII IIII					
Description	EALLOV		e ST1 regist				sed by 0:16l	bitAddr. The
Flags	This inst	ruction d	oes not mod	lify any S	STF register	flags.		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
Pipeline	This is a	two-cycl	e instruction	I.				
Example	MOVI MOVX NOP MOV3 MOV3	XI R5H, 32 ACC,	#0x1234 #0xABCD R5H A000), @ACC	; R5H[1 ; 1 Ali ; ACC = ; [0x00 ; 1 Cyc ; < M	1:16] = 0x1 5:0] = 0xA gnment Cycl 0x1234ABCD A000] = ACC le delay fo 0V32 *(0:16 A000] = 0xA	BCD e r MOV32 to bitAddr), l	.oc32 comple	
See also	MOV32	mem32, mem32, loc32, *((						



			ing i en	it negletel	•••••••				
Operands									
	ACC		28x accun	nulator					
	RaH		floating-po	oint source regi	ster (R0H to R	:7H)			
Opcode	LSW: 101 MSW: III		loc32 IIII IIII	I					
Description	floating-p	point reg		n move the ated by Ral		referenced	by mem32	to the	
	ACC = Ra								
Flags	No STF flags are affected.								
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	No	No	
Pipeline	While thi copying floating p MINF3 NOP MOV32 NOP	s is a sir a floating point inst 2 ROH,R1 @ACC,R	ngle-cycle g-point reg ruction, a . <sup>H</sup>	<pre>gister to a C single align ; Single-cy ; 1 alignmd ; Copy ROH ; Any inst;</pre>	additional p 28x register ment cycle ment cycle to ACC ruction	pipeline alig r. If the mov must be ad ction	nment is rec ve follows a s ded. For exa	single cycle	
			or example						
	NOP	2 R2H, F ACC, F		; < ADDF; ; 1 alignme ; copy R2H	delay for A 32 complete ent cycle into ACC, 2 completes	DDF32 to co s, R2H is v takes 2 cyc	valid cles		
Example		F32 ROH UI32 ROH P,		; Delay fo:	-	n instructi			
See also	MOV32 MOV32 MOV32	XARn, R							

# MOV32 ACC, RaH Move 32-bit Floating-Point Register Contents to ACC



# MOV32 loc32, \*(0:16bitAddr) Move 32-bit Value from Memory to loc32

Operands								
	loc32	destinatio	n location					
	0:16bitAddr	16-bit add	ress of the 32-	bit source valu	ue			
Opcode	LSW: 1011 111 MSW: IIII III		ſ					
Description	Copy the 32-bi	t value refei	enced by 0	:16bitAddr t	to the location	on indicated	by loc32.	
	[loc32] = [0:1	[6bitAddr]						
Flags	No STF flags a register zero (\$				gister, then t	the Z and N	flag in statu	S
	Flag TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified No	No	No	No	No	No	No	
Pipeline	This is a 2 cyc	le instructior	۱.					
Example	MOV @0 MOV @1	#0x0300 #0xFFFF #0x1111 CC, *(0xC000	; [0x00C ; [0x00C )) ; AL = [ ; 1 Cycl	000] = 0xFF 001] = 0x11 0x00C000], e delay for	111; AH = [0x000 MOV32 to 0	-	0×1111	
See also	MOV32 RaH, I MOV32 *(0:16 MOV32 STF, r MOVD32 RaH	oitAddr), loc nem32						



Instructions

#### www.ti.com

# MOV32 mem32, RaH Move 32-bit Floating-Point Register Contents to Memory

Operands												
	RaH	floating-point register (R0H to R7H)										
	mem32	points to the 32-bit de	stination memory									
Opcode	LSW: 1110 0010 MSW: 0000 0aaa	0000 0011 mem32										
Description	Move from mem [mem32] = RaH	ory to STF.										
	This instruction modifies the following flags in the STF register:											
Flags				-								
	Flag TF	ZI NI	ZF	NF	LUF	LVF						
	Modified No	No No	No	No	No	No						
	No flags affecte	d.										
Pipeline	This is a single-	cycle instruction.										
Example	-	ultiply and accumu	late operations									
Example	;		Tate Operations	•								
	-	.y: A = X0 * Y0 .y: B = X1 * Y1										
	<pre>; 3rd multiply: C = X2 * Y2 ; 4th multiply: D = X3 * Y3</pre>											
	-	Y: D = X3 * Y3 Y: E = X3 * Y3										
	; ; Result = A	+ B + C + D + E										
	MOV32 R0H,	*XAR4++	; R0H = X0									
	MOV32 R1H,	*XAR5++	; R1H = Y0									
		D011 D111	; R6H = A = 2									
	MPYF32 R6H,    MOV32 R0H,		; In paralle	L RUH = XI								
	MOV32 R1H,	*XAR5++	; R1H = Y1									
			; R7H = B = X1 * Y1									
	MPYF32 R7H,    MOV32 R0H,		; In paralle	L ROH = X2								
	MOV32 R1H,		; R1H = Y2									
			; R7H = A + E ; R6H = C = 2									
		R6H, R6H, R0H, R11										
	MOV32 R0H, MOV32 R1H,		; R1H = Y3									
			; R3H = (A +									
	MACF32 R7H,	R6H, R6H, R0H, R11	; R6H = D = ½ H ; In parallel									
	MOV32 R0H, MOV32 R1H,		; R1H = Y4									
			; R6H = E = >	(4 * Y4								
	MPYF32 R6H,		; in paralle		B + C) + D							
	ADDF32 R7H, NOP	R/H, RZH	; Wait for MB	PYF32    ADD	F32 to compl	ete						
	ADDF32 R7H,	R7H, R6H	; R7H = (A +	B + C + D)	+ E							
	NOP		; Wait for AI	DDF32 to com								
	MOV32 @Res		; Store the r	result								
See also	MOV32 *(0:16bi MOV32 mem32											



# MOV32 mem32, STF Move 32-bit STF Register to Memory

Operands											
	STF		floating-po	oint status regi	ster						
	mem32		points to t	he 32-bit desti	nation memory	/					
Opcode		LSW: 1110 0010 0000 0000 MSW: 0000 0000 mem32									
Description		Copy the floating-point status register, STF, to memory. [mem32] = STF									
Flags	This inst	This instruction modifies the following flags in the STF register:									
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No	No	No			
	No flags	affecte	d.								
Pipeline	This is a	single-	cycle instru	uction.							
Example 1		ZF32 R ZF32 R 732 R	OH, #2.0	; R1H = ; ZF = 0	2.0 (0x4000 3.0 (0x4040	00000) STF = 0x0000	00004				
Example 2	MOV3 MOVE MOVE CMPE MOV3 MOV3	732 R2 732 R3 732 R2 732 R3	н, к3н	; R2H = 3 ; R3H = 5 ; ZF = 0, T ; R3H = 3	TF in stack .0 (0x40400 .0 (0x40A00 NF = 1, ST .0 (0x40400 STF from s	0000) 0000) TF = 0x00000 0000)	0004				
See also	MOV32 MOV32 MOVST	*(0:16bi	RaH tAddr), loc	32							



MOV32 P, RaH	Move 32-bit Floating-Point Register Contents to P								
Operands									
	P RaH			uct register P point source reg	ister (R0H to F	R7H)			
Opcode	LSW: 1011 1111 loc32 MSW: IIII IIII IIII								
Description	Move th	e 32-bit	value in F	RaH to the 28	3x product r	egister P.			
Flags	No flags	affecte	d in floatir	ng-point unit.					
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	No	No	
Pipeline	While this is a single-cycle instruction, additional pipeline alignment is required when copying a floating-point register to a C28x register. If the move follows a single cycle floating point instruction, a single alignment cycle must be added. For example: MINF32 R0H,R1H ; Single-cycle instruction NOP ; 1 alignment cycle MOV32 @ACC,R0H ; Copy R0H to ACC NOP ; Any instruction								
			ws a 2 pip or examp		loating poir	nt instruction	i, then two a	lignment cycles	
	NOP NOP	32 R2H, 2 ACC,		; < ADDF ; 1 alignm ; copy R2H	delay for A 32 complete ent cycle into ACC, 2 completes	tion (2p) ADDF32 to co es, R2H is takes 1 cyo s, ACC is va	valid cle		
Example		ZF32 R0 DUI32 R0 2 P,		; Delay fo	ersion comp t cycle	0000 on instruct: plete, ROH v			
See also	MOV32 MOV32 MOV32	XARn, I	RaH						



#### MOV32 RaH, ACC Move the Contents of ACC to a 32-bit Floating-Point Register

Operands							
	RaH		floating-po	int destinatior	register (R0H	to R7H)	
	ACC		accumulat	or			
Opcode	LSW: 1011 MSW: III1	1 1101 I IIII					
Description	Move the		t value in A0	CC to the fl	oating-point	register Ra	ιH.
Flags	This instr	uction	does not m	odify any S	STF register	flags.	
	Flag	TF	ZI	NI	ZF	NF	
	Modified	No	No	No	No	No	
Pipeline	alignmen	t cycle	single-cycle es are requir egister. The	ed after an	y copy from	a standard	28

e-cycle instruction, additional pipeline alignment is required. Four e required after any copy from a standard 28x CPU register to a floating-point register. The four alignment cycles can be filled with any non-conflicting instructions except for the following: FRACF32, UI16TOF32, I16TOF32, F32TOUI32, and F32TOI32.

LUF

No

LVF

No

	MOV32 R0H,@ACC NOP NOP NOP NOP	; Copy ACC to R0H ; Wait 4 cycles ; Do not use FRACF32, UI16TOF32 ; I16TOF32, F32TOUI32 or F32TOI32 ; ; < R0H is valid
Example	MOV32 ROH, ACC NOP NOP NOP	; ACC = 512 H ; R0H = 512.0 (0x44000000)
See also	MOV32 RaH, P MOV32 RaH, XARn MOV32 RaH, XT	



#### Instructions

### MOV32 RaH, mem32 {, CNDF} Conditional 32-bit Move

#### Operands

RaH	floating-point destination register (R0H to R7H)
mem32	pointer to the 32-bit source memory location
CNDF	optional condition.

#### Opcode

LSW: 1110 0010 1010 CNDF MSW: 0000 0aaa mem32

Description

If the condition is true, then move the 32-bit value referenced by mem32 to the floating-point register indicated by RaH.

if (CNDF == TRUE) RaH = [mem32]

#### CNDF is one of the following conditions:

Encode (1)	CNDF	Description	STF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 AND NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF <sup>(2)</sup>	Unconditional with flag modification	None

<sup>(1)</sup> Values not shown are reserved.

<sup>(2)</sup> This is the default operation if no CNDF field is specified. This condition will allow the ZF, NF, ZI, and NI flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

This instruction modifies the following flags in the STF register:

Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
Modified	No	Yes	Yes	Yes	Yes	No	No	

if(CNDF == UNCF)
{
 NF = RaH(31);
 ZF = 0;
 if(RaH[30:23] == 0) { ZF = 1; NF = 0; }
 NI = RaH[31];
 ZI = 0;
 if(RaH[31:0] == 0) ZI = 1;
}
else No flags modified;

Pipeline

Flags

This is a single-cycle instruction.



Example	MOV         @0, #0x5555           MOV         @1, #0x5555           MOVIZF32         R3H, #7.0           MOVIZF32         R4H, #7.0           MAXF32         R3H, R4H	<pre>; DP = 0x0300 ; [0x00C000] = 0x5555 ; [0x00C001] = 0x5555 ; R3H = 7.0 (0x40E00000) ; R4H = 7.0 (0x40E00000) ; ZF = 1, NF = 0 ; R1H = 0x55555555</pre>
See also	MOV32 RaH, RbH{, CNDF}	

MOVD32 RaH, mem32



Instructions

Operands								
	RaH		floating-po	int register (R	0H to R7H)			
	Р		product ree	gister				
pcode	LSW: 101 MSW: III		loc32 IIII IIII					
escription	Move the RaH = P	e 32-bit	value in the	e product r	egister, P, t	o the floatin	g-point regis	ster RaH.
lags	This inst	ruction of	does not m	odify any S	STF register	flags.		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
lipeline	alignmer floating-j instructio	nt cycles point reg ons exce	s are requir gister. The	ed after an four alignm	y copy from	h a standard can be filled	28x CPU re with any no	quired. Four egister to a on-conflicting F32TOUI32,
ipeline	alignmer floating-j instructio and F32	nt cycles point reg ons exce TOI32.	s are requir gister. The ept for the f	ed after an four alignm ollowing: F	y copy from lent cycles ( RACF32, U	h a standard can be filled	28x CPU re with any no	egister to a on-conflicting
ipeline	alignmer floating-j instructio	nt cycles point reg ons exce TOI32.	s are requir gister. The ept for the f	ed after an four alignm following: F	y copy from ent cycles o RACF32, U	a standard can be filled II16TOF32,	28x CPU re with any no	egister to a on-conflicting
ipeline	alignmer floating-p instructio and F32 <sup>MOV32</sup> NOP NOP	nt cycles point reg ons exce TOI32.	s are requir gister. The ept for the f	red after an four alignm following: F ; Copy ; Wait ; Do n	y copy from ent cycles ( RACF32, U P to R0H 4 alignmer ot use FRAC	h a standard can be filled II16TOF32, nt cycles CF32, UI16TO	28x CPU re with any no I16TOF32, I	egister to a on-conflicting
ipeline	alignmer floating-r instructio and F32 <sup>MOV32</sup> NOP	nt cycles point reg ons exce TOI32.	s are requir gister. The ept for the f	red after an four alignm following: F ; Copy ; Wait ; Do n	y copy from ent cycles ( RACF32, U P to R0H 4 alignmer ot use FRAC	n a standard can be filled II16TOF32, nt cycles	28x CPU re with any no I16TOF32, I	egister to a on-conflicting
ipeline	alignmer floating-p instructio and F32 MOV32 NOP NOP NOP	nt cycles point reg ons exce TOI32.	s are requir gister. The ept for the f	red after an four alignm following: F ; Copy ; Wait ; Do n ; I16T ; ; <	y copy from ent cycles of RACF32, U P to R0H 4 alignmer ot use FRAG OF32, F32TC R0H is vali	n a standard can be filled II16TOF32, nt cycles CF32, UI16TC DUI32 or F3: id	28x CPU re with any no I16TOF32, I	egister to a on-conflicting
ipeline	alignmer floating-p instructio and F32 MOV32 NOP NOP NOP	nt cycles point reg ons exce TOI32.	s are requir gister. The ept for the f	red after an four alignm following: F ; Copy ; Wait ; Do n ; I16T ; ; <	y copy from ent cycles of RACF32, U P to R0H 4 alignmer ot use FRAG OF32, F32TC R0H is vali	n a standard can be filled l16TOF32, nt cycles CF32, UI16TC DUI32 or F33	28x CPU re with any no I16TOF32, I	egister to a on-conflicting
	alignmer floating-j instructio and F32 NOP NOP NOP NOP MOV MOV MOV MOV MOV MOV MOP NOP	PH, # PL, #	s are requir gister. The ept for the f @P 0x0000 0x0200	red after an four alignm following: F ; Copy ; Wait ; Do n ; I16T ; ; <	y copy from eent cycles of RACF32, U P to R0H 4 alignmer ot use FRAG OF32, F32TC R0H is vali ruction car	n a standard can be filled II16TOF32, nt cycles CF32, UI16TC DUI32 or F3: id	28x CPU re with any no I16TOF32, I	egister to a on-conflicting
Pipeline Example	alignmer floating-j instructio and F32 NOP NOP NOP NOP MOV MOV MOV MOV MOV MOV MOV MOV MOP NOP NOP	PH, # PL, #	s are requir gister. The ept for the f @P 0x0000 0x0200 P	red after an four alignm following: F ; Copy ; Wait ; Do n ; Il6T ; < ; Inst ; P =	y copy from eent cycles of RACF32, U P to R0H 4 alignmer ot use FRAG OF32, F32TC R0H is vali ruction car	h a standard can be filled U16TOF32, D16TOF32, U116TC DUI32 or F32 id h use R0H as	28x CPU re with any no I16TOF32, I	egister to a on-conflicting

# MOV32 RaH, P Move the Contents of P to a 32-bit Floating-Point Register



# MOV32 RaH, RbH {, CNDF} Conditional 32-bit Move

#### Operands

Flags

	RaH	floating-point des	stination register (R0H to R7H)	
	RbH	floating-point sou	urce register (R0H to R7H)	
	CNDF	optional conditio	n.	
Opcode	LSW: 1110 MSW: 0000			
Description	floating-poi	ition is true, then move int register indicated = TRUE) RaH = RbH	ve the 32-bit value reference by RaH.	ced by mem32 to the
	CNDF is o	ne of the following co	onditions:	
	Encode <sup>(1)</sup>	CNDF	Description	STF Flags Tested
	0000	NEQ	Not equal to zero	ZF == 0
	0001	EQ	Equal to zero	ZF == 1
	0010	GT	Greater than zero	ZF == 0 AND NF == 0
	0011	GEQ	Greater than or equal to zero	NF == 0
	0100	LT	Less than zero	NF == 1
	0101	LEQ	Less than or equal to zero	ZF == 1 AND NF == 1
	1010	TF	Test flag set	TF == 1
	1011	NTF	Test flag not set	TF == 0
	1100	LU	Latched underflow	LUF == 1
	1101	LV	Latched overflow	LVF == 1
	1110	UNC	Unconditional	None
	1111	UNCF <sup>(2)</sup>	Unconditional with flag modification	None

<sup>(1)</sup> Values not shown are reserved.

<sup>(2)</sup> This is the default operation if no CNDF field is specified. This condition will allow the ZF, NF, ZI, and NI flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

This instruction modifies the following flags in the STF register:

Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
Modified	No	Yes	Yes	Yes	Yes	No	No	

if(CNDF == UNCF)

{

}

NF = RaH(31); ZF = 0; if(RaH[30:23] == 0) {ZF = 1; NF = 0;} NI = RaH(31); ZI = 0; if(RaH[31:0] == 0) ZI = 1;

else No flags modified;

Pipeline	This is a single-cycle instruction.
Example	MOVIZF32R3H, #8.0; R3H = 8.0 (0x41000000)MOVIZF32R4H, #7.0; R4H = 7.0 (0x40E00000)MAXF32R3H, R4H; ZF = 0, NF = 0MOV32R1H, R3H, GT; R1H = 8.0 (0x41000000)
See also	MOV32 RaH, mem32{, CNDF}



# MOV32 RaH, XARn Move the Contents of XARn to a 32-bit Floating-Point Register

Operands							
	RaH	floating-po	oint register (R	0H to R7H)			
	XARn	auxiliary r	egister (XAR0	- XAR7)			
Opcode	LSW: 1011 1 MSW: IIII I		Ľ				
Description	Move the 32 RaH = XARn	2-bit value in th	e auxiliary i	egister XAF	Rn to the flo	ating point r	egister RaH.
Flags	This instruct	tion does not m	nodify any S	TF register	flags.		
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No	No	No	No	No	No	No
Pipeline	alignment c floating-poir	a single-cycle ycles are requi nt register. The except for the 32.	red after an four alignm	y copy from ent cycles o	a standard can be filled	28x CPU re with any no	egister to a
	NOP NOP NOP NOP	ROH,@XAR7	; Wait ; Do n ; I16T ; ; <	OF32, F32TC ROH is vali	nt cycles CF32, UI16TC DUI32 or F32 Ld	2TOI32	
	ADDF32	R2H,R1H,R0H	; Inst	ruction car	n use ROH as	s a source	
Example	MOV32 NOP NOP NOP NOP	XAR1, #0x0200 ROH, XAR1 32 ROH, ROH			0000)		
See also	MOV32 Rał MOV32 Rał MOV32 Rał	I, P					



perands								
peranus								
	RaH		floating-po	int register (R	OH to R7H)			
	ХТ		auxiliary re	egister (XAR0	- XAR7)			
pcode	LSW: 101 MSW: III		loc32 IIII IIII	:				
escription	Move the	e 32-bit	value in ter	mporary reg	gister, XT, to	o the floatin	g-point regis	ster RaH.
•	RaH = XI						01 0	
ags	This inst	ruction	does not m	odify any S	TF register	flags.		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
ipeline	alignmer floating-p	nt cycle point re	s are requir gister. The	red after an four alignm	y copy from ent cycles c	a standard an be filled		egister to a n-conflicting
	alignmer floating-j instructio	nt cycle point re pns exc	s are requir gister. The	red after an four alignm	y copy from ent cycles c	a standard an be filled	28x CPU re	egister to a n-conflicting
	alignmer floating-j instructio and F32 <sup>MOV32</sup> NOP NOP NOP NOP	nt cycle point re pns exc TOI32. R0H	s are requir gister. The ept for the f	red after an four alignm following: F ; Copy ; Wait ; Do n ; I16T ; ; <	y copy from ent cycles c RACF32, UI XT to R0H 4 alignmen ot use FRAC DF32, F32TO R0H is vali	a standard an be filled 16TOF32, t cycles F32, UI16TC UI32 or F32 d	28x CPU re with any no I16TOF32, F	egister to a n-conflicting
xample	alignmer floating- instructio and F32 MOV32 NOP NOP NOP NOP	2 R2H ZF32 F 2 2 2	s are requir gister. The ept for the f , xT ,R1H,R0H R6H, #5.0 KT, R6H	red after an four alignm following: F ; Copy ; Wait ; Do n ; I16T ; ; < ; Inst ; R6H = 5. ; 1 Alignm ; XT = 5.0	y copy from ent cycles c RACF32, U XT to ROH 4 alignmen ot use FRAC DF32, F32TO ROH is vali ruction can 0 (0x40A000	a standard an be filled 16TOF32, t cycles F32, UI16TC UI32 or F32 d use R0H as 00)	28x CPU re with any no I16TOF32, F	egister to a n-conflicting

# MOV32 RaH, XT Move the Contents of XT to a 32-bit Floating-Point Register



Instructions

www.ti.com

# MOV32 STF, mem32 Move 32-bit Value from Memory to the STF Register

Operands								
	STF		floating-po	pint unit status	register			
	mem32		pointer to	the 32-bit sour	ce memory loc	ation		
Opcode	LSW: 111 MSW: 000		1000 0000 mem32	0				
Description	Move fro	m mem	ory to the	floating-poir	it unit's state	us register S	STF.	
	STF = [m	em32]						
Flags	This inst	ruction	modifies th	ne following	flags in the	STF registe	r:	
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Restorin	g status	s register w	vill overwrite	all flags.			
Pipeline	This is a	single-	cycle instru	uction.				
Example 1	MOVW MOV MOV MOV3	@2, @3,	#0x020C #0x0000	; DP = 0x030 ; [0x00C002] ; [0x00C003] ; STF = 0x00	= 0x020C = 0x0000			
Example 2	MOV 3 MOVF MOVF CMPF MOV 3 MOV 3	232 R2 232 R3 232 R2 232 R3	2P++, STF 2H, #3.0 H, #5.0 2H, R3H H, R2H, L7 2F, *SP	; R2H = 3. ; R3H = 5. ; ZF = 0, T ; R3H = 3.	TF in stack 0 (0x40400) 0 (0x40A00) NF = 1, ST 0 (0x40400) STF from s	000) F = 0x00000 000)	004	
See also	MOV32 MOVST		, STF					



# MOV32 XARn, RaH Move 32-bit Floating-Point Register Contents to XARn

Operands	VAD-		00, 0,	ion register (V)				
	XARn			iary register (XA				
	RaH		floating-p	oint source reg	Ister (RUH to F	(7H)		
Opcode	LSW: 101 MSW: III		loc32 IIII III	I				
Description	Move the	e 32-bit	value fror	n the floating	g-point regis	ter RaH to	the auxiliary	register XARn.
	XARn = F	RaH						
Flags	No flags	affecte	d in floatir	ng-point unit.				
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
Pipeline	Copying floating MINF3 NOP	a floatir	ng-point restruction, a	e instruction, gister to a C a single align ; Single-c ; 1 alignm ; Copy ROH ; Any inst	28x registe ment cycle ycle instru ent cycle to ACC	r. If the mov must be ad	e follows a	single cycle
			ows a 2 pip For examp		loating poin	t instructior	n, then two a	lignment cycles
	ADDF3 NOP MOV32 NOP			; < ADDF ; 1 alignm ; copy R2H	delay for A 32 complete ent cycle into ACC, 2 completes	ion (2p) DDF32 to co s, R2H is v takes 1 cyc , ACC is va	valid cle	
Example		DUI32 R(	ОН, #2.5 ОН, ROH АRO, ROH	; Delay fo ; < Conv ; Alignmen	ersion comp	n instruct: Dete, ROH v		
See also	MOV32 MOV32 MOV32	P, RaH						



Operands								
	XT RaH		•	ry register	ictor (POH to F	)7U\		
	Kali		noating-p	point source reg		((1))		
Opcode	LSW: 101 MSW: III		loc32 IIII III	II				
Description	Move th		value in F	RaH to the te	mporary re	gister XT.		
Flags	No flags	affecte	d in floatir	ng-point unit.				
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
		ove follo			to ACC ruction	t instruction	i, then two a	lignment cycles
	ADDF3 NOP NOP		R1H, R0H	; 2 pipeli ; 1 cycle ; < ADDF ; 1 alignm ; copy R2H	delay for A 32 complete ent cycle into ACC, 2 completes	ion (2p) DDF32 to co s, R2H is v takes 1 cyc , ACC is va	valid cle	
Example	MOVIZ	ZF32 RC DUI32 RC	)H, #2.5 )H, ROH	; R0H = 2. ; Delay fo	5 = 0x40200 r conversic ersion comp	000 n instructi lete, ROH W		
	MOV32	2 XI	7, R0H	-	0x00000002			
See also	MOV32 MOV32 MOV32	P, RaH						

94



# MOVD32 RaH, mem32 Move 32-bit Value from Memory with Data Copy

Operatius								
	RaH		floating-poi	nt register (R	)H to R7H)			
	mem32		pointer to t	he 32-bit sour	ce memory loca	ation		
Opcode	LSW: 111 MSW: 000		0010 0011 mem32					
Description	Move the RaH.	32-bit	value refere	enced by m	em32 to the	e floating-po	int register ir	ndicated by
	RaH = [m [mem32+2		m32]					
Flags	This instr	uction r	modifies the	e following	flags in the S	STF register		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	Yes	Yes	Yes	Yes	No	No
	NI = RaH ZI = 0;	D:23] =: [31];	= 0){ ZF = 0) ZI = 1	1; NF = 0	; }			
Pipeline	This is a	single-c	cycle instru	ction.				
Example	MOVW MOV MOV MOVD	@2, @3,	#0x0000 #0x4110 , @2	; [0x00B00] ; R7H = 0x4	2] = 0x0000 3] = 0x4110	, [0x00B005	] = 0x4110	
See also	MOV32 F	RaH, me	em32 {,CNI	DF}				



Instructions

MOVF32 RaH, #32F	Load the	e 32-bits	s of a 32-bit	t Floating-P	oint Regis	ster		
Operands	translate MOVI	d by the z RaH, #		r MOVIZ and such that the				d operand is
	RaH		floating-point	t destination reg	gister (R0H to	R7H)		
	#32F		immediate flo	oat value repre	sented in floa	ting-point repre	esentation	
Opcode	LSW: 111 MSW: III			opcode of MC	OVIZ RaH, a	#16FHiHex)		
	LSW: 111 MSW: III			opcode of MO	OVXI RaH, :	<pre>#16FLoHex)</pre>		
Description	represen	tation. T	o specify th	s the immed e immediate 2 RaH, #32I	e value as a	a hex value		
	Load the	32-bits	of RaH with	the immedi	ate float va	lue represe	nted by #3	2F.
	accept a	float val ted as #	ue represer		ng-point re	presentation		embler will only 3.0 can only be
Flags	This inst	ruction m	nodifies the	following fla	gs in the S	TF register:		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
Pipeline	of the IEI convert M floating-p	EE 32-bi MOVF32 point forr	t floating-po into only M	oint format o OVIZ instrue are not zero	f #32F are ction. If the	zeros, then lower 16-b	the assem ts of the IE	
Example	MOVF	32 R1H	1, #3.0	; Assemble		s this inst	ruction as	
	MOVF	32 R2H	1, #0.0	; R2H = 0. ; Assemble			ruction as	
	MOVF	32 R3H	1, #12.265	; Assemble ; MOVIZ R3	2.625 (0x4)	s this inst 4	ruction as	
See also	Moviz F Movxi F Movi32 Movizf:	RaH, #16 RaH, #3	FLoHex 2FHex					

# MOVF32 RaH, #32F Load the 32-bits of a 32-bit Floating-Point Register



# MOVI32 RaH, #32FHex Load the 32-bits of a 32-bit Floating-Point Register with the immediate

				•	•		
Operands		ion is an alias y the assembl					l operand is
		aH, #16FHiHex aH, #16FLoHex					
	RaH	floating-p	oint register (R0	H to R7H)			
	#32FHex	A 32-bit i	mmediate value	that represents	an IEEE 32-b	oit floating-point	value.
Opcode	LSW: 1110 1	000 0000 0III	(opcode of	MOVIZ RaH, ‡	16FHiHex)		
		III IIII Iaaa 000 0000 1III		MOVXI RaH, ‡	16FLoHex)		
	MSW: IIII I	III IIII Iaaa					
Description		nstruction only alue with a flo					
	Load the 32	-bits of RaH w	vith the imme	diate 32-bit h	nex value r	epresented I	by #32Fhex.
	value of a flo	can only be re	umber. The a	ssembler wil	l only acce	pt a hex imr	nediate value.
lags	This instruct	ion modifies t	ne following t	lags in the S	TE register	··	
	Flag TF	ZI	NI	ZF	NF	LUF	LVF
	Modified No		No	No	No	No	No
Pipeline	16-bits of #3 instruction. I	on #32FHex, t 82FHex are ze f the lower 16 a MOVIZ and	ros, then as bits of #32F	sembler will o Hex are not z	onvert MO	VI32 to the	MOVIZ
Example	MOVI32	R1H, #0x404	; As	H = 0x404000 sembler conv VIZ R1H, #02	verts this	instruction	n as
	MOVI32	R2H, #0x000	00000 ; R2 ; As	$2H = 0 \times 000000$ sembler conv VIZ R2H, #02	)00 verts this	instruction	1 as
	MOVI32	R3H, #0x400	04001 ; R3 ; As ; M0	BH = 0x400040 sembler conv VIZ R3H, $\#0_2$ VXI R3H, $\#0_3$	001 verts this 4000	instruction	1 as
	MOVI32	R4H, #0x000	04040 ; R4 ; As ; M0	H = 0x000040 sembler conv VIZ R4H, #02 VXI R4H, #03	040 verts this 00000	instruction	n as
See also	MOVIZ RaH MOVXI RaH	, #16FHiHex					



# MOVIZ RaH, #16FHiHex Load the Upper 16-bits of a 32-bit Floating-Point Register

Operatios									
	RaH		floating-po	int register (R	OH to R7H)				
	#16FHiHe>	K					er 16-bits of an re assumed to b		
Opcode	LSW: 1110 MSW: IIII		0000 0III IIII Iaaa						
Description		e value					ate operand IOVIZF32 ps	. To specify the seudo	Э
	Load the 16-bits of		16-bits of F	RaH with the	e immediate	value #16F	HiHex and c	lear the low	
	32-bit floa assemble	ating-po er will o	bint value. nly accept	The low 16-	bits of the m diate value.	nantissa are		f an IEEE be all 0. The e represented	
	lowest 16 (0x40800	6-bits of 000), 0 a float n. 5] = #1	the mantis 1.5 (0x3F00) ing-point re	ssa are 0. S 00000), and	ome examp -1.5 (0xBFC	les are 2.0 200000). If a	(0x4000000) a constant re		
Flags	This instr	uction i	modifies the	e following	flags in the	STF registe	r:		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	-
	Modified	No	No	No	No	No	No	No	_
Pipeline	This is a	single-	cycle instru	ction.					
Example	; Load RC MOVI2		-1.5 (0xB 0H, #0xBFC		= 0xBFC0000	00			
		)H with Z R		1593 (0x40 9 ; ROH		00			
See also	MOVIZF3 MOVXI R								



# MOVIZF32 RaH, #16FHi Load the Upper 16-bits of a 32-bit Floating-Point Register

Operands	_							
	RaH		floating-point	register (R	)H to R7H)			
	#16FHi					ts the upper 16-b the mantissa are		
			01					
Opcode	LSW: 111 MSW: III		0000 0III IIII Iaaa					
Description	Load the 16-bits o		6-bits of Ra	H with the	e value repr	esented by #	16FHi and	clear the low
	floating-µ addressi are 0. So -1.5 (0xE	point val ing mode ome exa 3FC0000	ue. The low e is most use mples are 2.	16-bits of eful for co 0 (0x400 can be sp	the mantise nstants whe 00000), 4.0	s the upper 1 sa are assum ere the lowest (0x40800000 ex or float. Th	ed to be al 16-bits of ), 0.5 (0x3	l 0. This the mantissa F000000), and
	MOVIZF encodes	32 the a it into a as MOV	ssembler wi MOVIZ instr /IZ RaH, 0xB	Il accept e ruction. F	either a hex	iHex instructi or float as the MOVIZF32 F	e immediat	e value and
Flags	This inst	ruction r		following	0	STF register:		
Flags	Flag	ruction r	ZI	NI	ZF	NF	LUF	LVF
Flags		ruction r		•	0	0	LUF No	LVF No
Flags Pipeline	Flag Modified	TF No	ZI	NI No	ZF	NF		
-	Flag Modified This is a MOVI MOVI MOVI ; ; Load R ; ; Load R ; ; ; Load R	<b>TF</b> No <b>Single-C</b> ZF32 R01 ZF32 R11 ZF32 R11 ZF32 R31 ZF32 R31 ZF32 R41 R5H with ZF32 R51 R0H with ZF32 R51	<b>ZI</b> No cycle instruct H, #3.0 H, #1.0 H, #2.5 H, #-5.5 H, #0xC0B0 pi = 3.1415 H, #3.141593 a more accu DH, #0x4049	NI No ion. ; R0H ; R1H ; R1H ; R2H ; R3H ; R4H ; R4H ; R5H ; R5H ; R5H	ZF No = 3.0 = 0 = 1.0 = 0 = 2.5 = 0 = -5.5 = 0 = -5.5 = 0 490000) = 3.14062 = 3.141593 = 0x404900	NF           No           x40400000           x3F800000           x40200000           xC0B00000           xC0B00000           5 (0x40490000)           (0x40490FDB)           00	No	
Pipeline	Flag Modified This is a MOVI MOVI MOVI MOVI ; ; Load R ; MOVI ; Load R ; MOVI MOVIZ	TF         No         2ZF32         2ZF32         2ZF32         2ZF32         2ZF32         2ZF32         2ZF32         R5H         R5H         R0H         R1         R2F32         R3H         R4H	<b>ZI</b> No cycle instruct H, #3.0 H, #1.0 H, #2.5 H, #0xC0B0 pi = 3.1415 H, #3.141593 a more accu DH, #0x4049 DH, #0x0FDB	NI No ion. ; R0H ; R1H ; R1H ; R2H ; R3H ; R4H ; R4H ; R5H ; R5H ; R5H	ZF No = 3.0 = 0 = 1.0 = 0 = 2.5 = 0 = -5.5 = 0 = -5.5 = 0 1900000) = 3.14062 = 3.141593	NF           No           x40400000           x3F800000           x40200000           xC0B00000           xC0B00000           5 (0x40490000)           (0x40490FDB)           00	No	



MOVST0 FLAG Load Selected STF Flags into ST0 Operands FLAG Selected flag Opcode LSW: 1010 1101 FFFF FFFF Load selected flags from the STF register into the ST0 register of the 28x CPU where Description FLAG is one or more of TF, CI, ZI, ZF, NI, NF, LUF or LVF. The specified flag maps to the ST0 register as follows: Set OV = 1 if LVF or LUF is set. Otherwise clear OV. Set N = 1 if NF or NI is set. Otherwise clear N. . Set Z = 1 if ZF or ZI is set. Otherwise clear Z. Set C = 1 if TF is set. Otherwise clear C. Set TC = 1 if TF is set. Otherwise clear TF. If any STF flag is not specified, then the corresponding ST0 register bit is not modified. Restrictions Do not use the MOVST0 instruction in the delay slots for pipelined operations. Doing so can yield invalid results. To avoid this, the proper number of NOPs or non-pipelined instructions must be inserted before the MOVST0 operation. The following is INVALID MPYF32 R2H, R1H, R0H ; 2 pipeline-cycle instruction (2p) MOVSTO TF ; INVALID, do not use MOVSTO in a delay slot ; The following is VALID MPYF32 R2H, R1H, R0H ; 2 pipeline-cycle instruction (2p) NOP 1 delay cycle, R2H updated after this instruction ; MOVSTO TF ; VALID This instruction modifies the following flags in the STF register: Flags TF ΖI NI ZF NF LUF LVF Flag Modified No No No No No Yes Yes When the flags are moved to the C28x ST0 register, the LUF or LVF flags are automatically cleared if selected. This is a single-cycle instruction. Pipeline Example Program flow is controlled by C28x instructions that read status flags in the status register 0 (ST0). If a decision needs to be made based on a floating-point operation, the information in the STF register needs to be loaded into ST0 flags (Z,N,OV,TC,C) so that the appropriate branch conditional instruction can be executed. The MOVST0 FLAG instruction is used to load the current value of specified STF flags into the respective bits of ST0. When this instruction executes, it will also clear the latched overflow and underflow flags if those flags are specified. Loop: MOV32 R0H, \*XAR4++ MOV32 R1H, \*XAR3++ CMPF32 R1H, R0H MOVSTO ZF, NF Loop, GT ; Loop if (R1H > R0H) BF See also MOV32 mem32, STF MOV32 STF, mem32



# MOVXI RaH, #16FLoHex Move Immediate to the Low 16-bits of a Floating-Point Register

operands									
	Ra		floating-po	oint register (R	0H to R7H)				
	#16FLoHe	ex			value that repre upper 16-bits v		er 16-bits of an I ified.	EEE 32-bit	
Opcode	LSW: 111 MSW: III		0000 1III IIII Iaaa	_					
Description	represer RaH will	nts the l not be	ower 16-bi modified. I	ts of an IEE MOVXI can	E 32-bit floa	ting-point va d with the M	Hex. #16FL alue. The up IOVIZ or MC	per 16-bits of	of
	RaH[15:0 RaH[31:1								
Flags									
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	No	No	
Pipeline	This is a	single-	cycle instru	uction.					
Example	; Load R MOVI MOVX	Z I	n pi = 3.14 ROH,#0x4049 ROH,#0x0FDH		490FDB) = 0x404900 = 0x40490F				
See also			l6FHiHex I, #16FHi						



# MPYF32 RaH, RbH, RcH 32-bit Floating-Point Multiply

Operands										
	RaH		floating-p	oint destinatior	register (R0H	to R7H)				
	RbH		floating-p	oint source reg	ister (R0H to F	R7H)				
	RcH		floating-p	oint source reg	ister (R0H to F	R7H)				
Opcode	LSW: 1110 MSW: 0000		0000 000 ccbb baa							
Description	Multiply t <sub>RaH</sub> = RbI			o floating-po	oint register	6.				
Flags	This instruction modifies the following flags in the STF register:.									
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No	Yes	Yes		
Pipeline	• LVF = This is a MPYF: NOP	= 1 if M 2 pipel 32 Raf uction i	PYF32 ge ine cycle ( <sup>I, RbH, Rc</sup> in the dela	; 1 cy ; <	overflow cor on. That is: peline cycl cle delay c MPYF32 comp	ndition. es (2p) or non-confi oletes, RaH	-	eruction er or use RaH		
Example	Calculate MOVL MOV32 MOVL MOV32 MPYF32 MOVL	Y = A XAR4, R0H, XAR4, R1H, 2 R0H, F XAR4,	* B: #A *XAR4 ; # B *XAR4 ; 21H,R0H ; #Y ;	Load ROH w Load R1H w Multiply A <mpyf32< th=""><th>ith B * B complete</th><th></th><th></th><th></th></mpyf32<>	ith B * B complete					
See also	MPYF32 MPYF32 MPYF32 MPYF32	RaH, # RaH, F RdH, F RdH, F RaH, F	#16FHi, Rt RbH, RcH ReH, RfH   ReH, RfH   RbH, RcH	Save the r H    ADDF32 F   MOV32 Ra   MOV32 m    SUBF32 F , ReH, RfH	RdH, ReH, I aH, mem32 em32, RaH RdH, ReH, F	RfH				



# MPYF32 RaH, #16FHi, RbH 32-bit Floating-Point Multiply

Operands						
	RaH	floating-point des	stination register (R0H	to R7H)		
	#16FHi		te value that represen ue. The low 16-bits of			
	RcH	floating-point sou	irce register (R0H to R	7H)		
Opcode	LSW: 1110 100 MSW: IIII III					
Description		with the floating-pone addition in RaH		nted by the	immediate c	perand. Store
	floating-point of most useful fo Some exampl (0xBFC00000 That is, the va RaH = RbH * #	6-bit immediate va value. The low 16- or representing cor es are 2.0 (0x4000 ). The assembler alue -1.5 can be re el6FHi:0 on can also be writ	bits of the mantise istants where the 00000), 4.0 (0x408 will accept either a presented as #-1.	sa are assu lowest 16-b 300000), 0.5 a hex or floa 5 or #0xBF0	med to be a its of the ma 5 (0x3F0000 it as the imn C0.	ll 0. #16FHi is antissa are 0. 00), and -1.5
Flags		n modifies the foll				
Tidgs	Flag TF			NF	LUF	LVF
	Modified No	No N	lo No	No	Yes	Yes
Pipeline	<ul> <li>LVF = 1 if</li> <li>This is a 2 pip</li> </ul>		es an overflow con struction. That is:	<b>dition.</b> es (2p) r non-confl		ruction
	NOP			icces, itali	apaacca	
	Any instruction as a source o	n in the delay slot perand.	must not use Ra⊢	l as a destir	nation regist	er or use RaH
Example 1	MOVIZF32 MPYF32 MOVL	R3H, #2.0 R4H, #3.0, R3H XAR1, #0xB006	<pre>; R3H = 2.0 (0x4 ; R4H = 3.0 * R3 ; &lt; Non confli ; &lt; MPYF32 com</pre>	H cting instr		000000)
	MOV32	*XAR1, R4H	; Save the resul	t in memory	/ location (	xB006
Example 2		re example but #16 R3H, #2.0 R4H, #0x4040, R3	5FHi is represent ; R3H = 2.0 ( 3H ; R4H = 0x404 ; 3.0 is repr ; IEEE 754 32	0x40000000) 0 * R3H esented as	0x40400000	in
	MOVL	XAR1, #0xB006	; < Non con ; < MPYF32	flicting in complete, R	struction 24H = 6.0 (0	
	MOV32	*XAR1, R4H	; Save the re	sult in mem	ory locatio	on 0xB006
See also	MPYF32 RaH	, RbH, #16FHi  , RbH, RcH  , RbH, RcH    ADI	DF32 RdH, ReH, F	RfH		

# MPYF32 RaH, RbH, #16FHi 32-bit Floating-Point Multiply

Operands								
	RaH	floating-point dest	nation register (R0H	ster (R0H to R7H)				
	RbH	floating-point sour	ce register (R0H to F	R7H)				
	#16FHi		e value that represer e. The low 16-bits of					
Incodo	TON: 1110 1000	0177 7777						
pcode	LSW: 1110 1000 MSW: IIII IIII							
escription		rith the floating-poi e addition in RaH.	nt value represe	nted by the	immediate o	perand. Store		
	floating-point v most useful for Some example (0xBFC00000)	-bit immediate valuation alue. The low 16-bit representing consets are 2.0 (0x4000). The assembler ware -1.5 can be rep	its of the mantis stants where the 0000), 4.0 (0x40 ill accept either a	sa are assu lowest 16-b 800000), 0.5 a hex or floa	med to be a hits of the ma 5 (0x3F0000 at as the imm	ll 0. #16FHi is Intissa are 0. 00), and -1.5		
		n can also be write						
lags		modifies the follo		•				
	Flag TF Modified No	ZI NI No No	ZF No	NF No	LUF Yes	LVF Yes		
Pipeline	This is a 2 pipe	/IPYF32 generates eline cycle (2p) ins h, rbh, #16Fhi ;	truction. That is:					
	NOP		1 cycle delay c < MPYF32 comp			ruction		
	Any instruction as a source op	in the delay slot n erand.	nust not use Rał	l as a destir	nation registe	er or use RaH		
Example 1		R4H, R3H, #3.0 XAR1, #0xB008		0 .cting instr		C00000)		
	MOLIZO			-				
	MOV32	*XAR1, R4H	Save the resul	.c III memory	y location U	xB008		
xample 2	;Same as above MOVIZF32	e example but #16B	THI IS REPRESENT ; R3H = 2.0 ( ) ; R4H = R3H * ; 3.0 IS REPR	ed in Hex 0x40000000) 0x4040 cesented as	) 0x40400000			
xample 2	;Same as above MOVIZF32 MPYF32	e example but #16E R3H, #2.0	'Hi is represent ; R3H = 2.0 ( ) ; R4H = R3H *	ed in Hex 0x40000000) 0x4040 esented as 2-bit format flicting ir	) 0x40400000 c istruction	in		
xample 2	;Same as above MOVIZF32 MPYF32	e example but #16F R3H, #2.0 R4H, R3H, #0x404(	<pre>Hi is represent     ; R3H = 2.0 (     ; R4H = R3H *     ; 3.0 is repr     ; IEEE 754 32     ; &lt; Non cor </pre>	ed in Hex 0x40000000) 0x4040 esented as 2-bit format flicting in complete, F	) 0x40400000 - struction R4H = 6.0 (0	in x40C00000)		



### MPYF32 RaH, RbH, RcH ADDF32 RdH, ReH, RfH 32-bit Floating-Point Multiply with Parallel Add

Operands								
	RaH			t destination rebeat the same in		YF32 (R0H to H	R7H)	
	RbH		floating-poin	t source regist	er for MPYF3	2 (R0H to R7H	I)	
	RcH		floating-poin	t source regist	er for MPYF3	2 (R0H to R7H	I)	
	RdH			t destination robe the same i		DF32 (R0H to H	R7H)	
	ReH		floating-poin	t source regist	er for ADDF3	2 (R0H to R7H	)	
	RfH		floating-poin	t source regist	er for ADDF3	2 (R0H to R7H	)	
Opcode	LSW: 111 MSW: fee		0100 00ff ccbb baaa					
Description	Multiply the contents of two floating-point registers with parallel addition of two registers. RaH = RbH * RcH RdH = ReH + RfH							
	This inst	ruction o	an also be	written as:				
	MACF32 F	RaH, RbH	, RcH, RdH,	ReH, RfH				
Restrictions	The destination register for the MPYF32 and the ADDF32 must be unique. That is, RaH cannot be the same register as RdH.						e. That is, RaH	
Flags	This instruction modifies the following flags in the STF register:.							
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	Yes	Yes
Pipeline	LUF     LVF     Doth MF     MPYH     I ADDE     NOP     NOP     NOP	= 1 if MF = 1 if MF PYF32 ar 732 RaH 732 RdH	PYF32 or AE ad ADDF32 , RbH, RcH , ReH, RfH	DDF32 geno DDF32 geno take 2 pipe ; 2 pipel ; 2 pipel ; 1 cycle ; < MPY	erates an u erates an o line cycles line cycles ine cycles delay or F32, ADDF3	s (2p) non-conflic 32 complete,	dition. : cting instr , RaH, RdH	

a source operand.



Evenue								
Example	; Perform 5 multiply and accumulate operations:							
	; 1st multiply: A = X0 * Y0							
	; 2nd multiply: B = X1 * Y1							
	; 3rd multiply: C = X2 * Y2							
	; 4th multiply: D = X3 * Y3							
	; 5th multiply: $E = X3 * Y3$ .							
	; Result = A + B + C + D + E							
	MOV32 R0H, *XAR4++ ; R0H = X0							
	MOV32 R1H, *XAR5++ ; R1H = Y0							
	; R2H = A = X0 * Y0							
	MPYF32 R2H, R0H, R1H ; In parallel R0H = X1							
	MOV32 R0H, *XAR4++ MOV32 R1H, *XAR5++ ; R1H = Y1							
	; R3H = B = X1 * Y1							
	MPYF32 R3H, R0H, R1H ; In parallel R0H = X2							
	MOV32 R0H, *XAR4++							
	MOV32 R1H, *XAR5++ ; R1H = Y2							
	; R3H = A + B							
	; R2H = C = X2 * Y2							
	MACF32 R3H, R2H, R2H, R0H, R1H ; In parallel R0H = X3							
	MOV32 R0H, *XAR4++							
	MOV32 R1H, *XAR5++ ; R1H = Y3							
	; R3H = (A + B) + C							
	; R2H = D = X3 * Y3							
	MACF32 R3H, R2H, R2H, R0H, R1H ; In parallel R0H = X4							
	MOV32 R1H, *XAR5 ; R1H = Y4							
	; R2H = E = X4 * Y4							
	MPYF32 R2H, R0H, R1H ; in parallel R3H = $(A + B + C) + D$							
	ADDF32 R3H, R3H, R2H NOP ; Wait for MPYF32    ADDF32 to complete							
	ADDF32 R3H, R3H, R2H ; $R3H = (A + B + C + D) + E$							
	NOP ; Wait for ADDF32 to complete							
	MOV32 @Result, R3H ; Store the result							
See also	MACF32 R3H, R2H, RdH, ReH, RfH							
	MACF32 R3H, R2H, RdH, ReH, RfH    MOV32 RaH, mem32							
	MACF32 R7H, R3H, mem32, *XAR7++							
	MACF32 R7H, R6H, RdH, ReH, RfH							
	MACF32 R7H, R6H, RdH, ReH, RfH    MOV32 RaH, mem32							



### MPYF32 RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Multiply with Parallel Move

Operands								
	RdH			egister for the egister as Ral		H to R7H)		_
	ReH	floating-point	source regist	er for the MP	YF32 (R0H to	R7H)		
	RfH	floating-point source register for the MPYF32 (R0H to R7H)						
	RaH floating-point destination register for the MOV32 (R0H to R7H) RaH cannot be the same register as RdH							
	mem32	pointer to a 3	32-bit memory	location. This	s will be the so	ource of the MC	DV32.	
Opcode	LSW: 1110 0011 MSW: eedd daaa							
Description	Multiply the cor RdH = ReH * Rf RaH = [mem32]		loating-poir	nt registers	and load a	nother.		
Restrictions	The destination register for the MPYF32 and the MOV32 must be unique. That is, RaH cannot be the same register as RdH.							
Flags	This instruction modifies the following flags in the STF register:.							
	Flag TF	ZI	NI	ZF	NF	LUF	LVF	
	Modified No	Yes	Yes	Yes	Yes	Yes	Yes	
	<ul> <li>The STF register flags are modified as follows:</li> <li>LUF = 1 if MPYF32 generates an underflow condition.</li> <li>LVF = 1 if MPYF32 generates an overflow condition.</li> </ul>							
	The MOV32 Instruction will set the NF, ZF, NI and ZI flags as follows:							
	<pre>NF = RaH(31); ZF = 0; if(RaH(30:23) == 0) { ZF = 1; NF = 0; } NI = RaH(31); ZI = 0; if(RaH(31:0) == 0) ZI = 1;</pre>							
Pipeline	MPYF32 takes 2 pipeline-cycles (2p) and MOV32 takes a single cycle. That is:							
	MPYF32 RdH, ReH, RfH ; 2 pipeline cycles (2p)    MOV32 RaH, mem32 ; 1 cycle ; < MOV32 completes, RaH updated							
	NOP	; 1 cycle delay or non-conflicting instruction ; < MPYF32 completes, RdH updated					ruction	
	1101							

Any instruction in the delay slot must not use RdH as a destination register or as a source operand.



Example	Calculate $Y = M1^*X1 + B1$ . This example assumes that M1, X1, B1 and Y1 are all on the same data page.						
	MOVW       DP, #M1       ; Load the data page         MOV32       R0H,@M1       ; Load R0H with M1         MOV32       R1H,@X1       ; Load R1H with X1         MPYF32       R1H,R1H,R0H       ; Multiply M1*X1            MOV32       R0H,@B1       ; and in parallel load R0H with B1         ; <       MOV32 complete         NOP       ; Wait 1 cycle for MPYF32 to complete						
	; < MPYF32 complete ADDF32 R1H,R1H,R0H ; Add M*X1 to B1 and store in R1H NOP ; Wait 1 cycle for ADDF32 to complete ; < ADDF32 complete ; Store the recent						
	MOV32 @Y1,R1H ; Store the result Calculate Y = (A * B) * C:						
	<pre>MOVL XAR4, #A MOV22 ROH, *XAR4 ; Load ROH with A MOVL XAR4, #B MOV22 R1H, *XAR4 ; Load R1H with B MOVL XAR4, #C MPYF32 R1H,R1H,R0H ; Calculate R1H = A * B    MOV32 ROH, *XAR4 ; and in parallel load R2H with C ; &lt; MOV32 complete MOVL XAR4, #Y ; &lt; MPYF32 complete MPYF32 R2H,R1H,R0H ; Calculate Y = (A * B) * C NOP ; Wait 1 cycle for MPYF32 to complete</pre>						
	MOV32 *XAR4,R2H						
See also	MPYF32 RdH, ReH, RfH    MOV32 mem32, RaH MACF32 R3H, R2H, RdH, ReH, RfH    MOV32 RaH, mem32 MACF32 R7H, R6H, RdH, ReH, RfH    MOV32 RaH, mem32 MACF32 R7H, R3H, mem32, *XAR7++						



### MPYF32 RdH, ReH, RfH MOV32 mem32, RaH 32-bit Floating-Point Multiply with Parallel Move

Operands											
	RdH	RdH floating-point destination register for the MPYF32 (R0H to R7H)									
	ReH	ReH floating-point source register for the MPYF32 (R0H to R7H)									
	RfH floating-point source register for the MPYF32 (R0H to R7H)										
	mem32	pointer to a 32	2-bit memory l	ocation. Th	is will be the d	estination of th	e MOV32.				
	RaH	floating-point s	source registe	er for the MC	DV32 (R0H to	R7H)					
								-			
Opcode	LSW: 1110 000 MSW: eedd daa										
Description	Multiply the constraints RdH = ReH * F [mem32] = Raf		pating-point	t registers	s and move	from memo	ry to register.				
Flags	This instruction modifies the following flags in the STF register:										
	Flag TF	ZI	NI	ZF	NF	LUF	LVF	_			
	Modified No	No	No	No	No	Yes	Yes	_			
Pipeline	MPYF32 take MPYF32 F	<pre>MPYF32 generates an overflow condition. s 2 pipeline-cycles (2p) and MOV32 takes a single cycle. That is: dH, ReH, RfH ; 2 pipeline cycles (2p) em32, RaH ; 1 cycle ; &lt; MOV32 completes, mem32 updated ; 1 cycle delay or non-conflicting instruction ; &lt; MPYF32 completes, RdH updated</pre>									
	Any instructio source opera	n in the delay slond.	ot must not	use Rd⊦	l as a desti	nation regist	er or as a				
Example	MPYF32	XAR1, #0xC003 R3H, #2.0 R3H, R3H, #5.0 R1H, #5.0 R3H, R1H, R3H *XAR1, R3H	; R3H = ; R3H = ; R1H = ; < ME ; R3H = ; and ir	2.0 (0x4 R3H * 5. 5.0 (0x4 PYF32 com R1H * R3 paralle	0 0A00000) plete, R3H H						
	NOP		-	-	for MPYF32	to complete					
See also	MACF32 R3F MACF32 R7F	I, ReH, RfH    M0 I, R2H, RdH, Re I, R6H, RdH, Re I, R3H, mem32,	H, RfH    M H, RfH    M	10V32 Ra							

MPYF32 RaH, RbH, RcH	
SUBF32 RdH, ReH, RfH	32-bit Floating-Point Multiply with Parallel Subtract

Operands										
	RaH		floating-point of RaH cannot be			32 (R0H to R	7H)			
	RbH		floating-point source register for MPYF32 (R0H to R7H)							
	RcH		floating-point source register for MPYF32 (R0H to R7H)							
	RdH		floating-point of RdH cannot be			32 (R0H to R7	7H)			
	ReH		floating-point s	source register	for SUBF32 (	R0H to R7H)				
	RfH		floating-point s	source register	for SUBF32 (	R0H to R7H)				
Opcode	LSW: 1110 MSW: feee									
Description	Multiply the registers.		contents of two floating-point registers with parallel subtraction of two							
	RdH = ReH	H - RfH								
Restrictions			egister for the		and the SUI	BF32 must	be unique	. That is, RaH		
Flags This instruction modifies the following flags in the STF register:.										
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No	Yes	Yes		
Pipeline	<ul> <li>LUF =</li> <li>LVF =</li> <li>MPYF32</li> <li>MPYF3</li> </ul>	= 1 if MP' = 1 if MP' and SUE	flags are mo YF32 or SUE YF32 or SUE BF32 both ta RbH, RcH ReH, RfH	3F32 genera 3F32 genera ke 2 pipelin ; 2 pipelin ; 2 pipelin	ates an unc ates an ove e-cycles (2 ne cycles ( ne cycles (	rflow condit p). That is:	tion.	uction		
	NOD			-	-	complete.				
	NOP Any instru a source		•	ot must not	use RaH or	RdH as a	destinatior	n register or as		
Example		ZF32 R51 32 R61	H, #5.0 H, #3.0 H, R4H, R5H H, R4H, R5H	; R5H = 3 ; R6H = R ; R7H = R ; 1 cycle ; < MPY	.0 (0x40400 4H * R5H 4H - R5H delay for F32    SUBB		e,	-		
See also		RdH, Re	H, RcH H, RfH    M0 H, RfH    M0							



#### Operands

RaH	floating-point destination register (R0H to R7H)
RbH	floating-point source register (R0H to R7H)
CNDF	condition tested

#### Opcode

Description

	 0110 0000	 	
if ( else	== tru	aH = - aH = R	

#### CNDF is one of the following conditions:

Encode (1)	CNDF	Description	STF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 AND NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF <sup>(2)</sup>	Unconditional with flag modification	None

<sup>(1)</sup> Values not shown are reserved.

<sup>(2)</sup> This is the default operation if no CNDF field is specified. This condition will allow the ZF, NF, ZI, and NI flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.
 This instruction modifies the following flags in the STF register:

### Flags

Flag	TF	ZI	NI	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes	No	No

Pipeline	This is a single-cycle instruction.
Example	MOVIZF32R0H, #5.0; R0H = 5.0 (0x40A00000)MOVIZF32R1H, #4.0; R1H = 4.0 (0x40800000)MOVIZF32R2H, #-1.5; R2H = -1.5 (0xBFC00000)
	MPYF32 R4H, R1H, R2H ; R4H = -6.0 MPYF32 R5H, R0H, R1H ; R5H = 20.0 ; < R4H valid
	CMPF32 R4H, #0.0 ; NF = 1 ; < R5H valid NEGF32 R4H, R4H, LT ; if NF = 1, R4H = 6.0 CMPF32 R5H, #0.0 ; NF = 0
	NEGF32 R5H, R5H, GEQ ; if NF = 0, R4H = -20.0

See also

ABSF32 RaH, RbH



POP RB Pop the RB Register from the Stack Operands RΒ repeat block register Opcode LSW: 1111 1111 1111 0001 Restore the RB register from stack. If a high-priority interrupt contains a RPTB Description instruction, then the RB register must be stored on the stack before the RPTB block and restored after the RTPB block. In a low-priority interrupt RB must always be saved and restored. This save and restore must occur when interrupts are disabled. Flags This instruction does not affect any flags floating-point Unit: Flag TF ΖI NI ZF NF LUF LVF Modified No No No No No No No This is a single-cycle instruction. **Pipeline** Example A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not have to save the RB register. ; Repeat Block within a High-Priority Interrupt (Non-Interruptible) \_Interrupt: ; RAS = RA, RA = 0PUSH RB ; Save RB register only if a RPTB block is used in the ISR . . . RPTB #BlockEnd, AL ; Execute the block AL+1 times . . . . . . BlockEnd ; End of block to be repeated . . . ; Restore RB register POP RB IRET ; RA = RAS, RAS = 0A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled. ; Repeat Block within a Low-Priority Interrupt (Interruptible) \_Interrupt: ; RAS = RA, RA = 0. . . PUSH RB ; Always save RB register . . . CLRC INTM ; Enable interrupts only after saving RB . . . ; ISR may or may not include a RPTB block . . . . . . SETC INTM ; Disable interrupts before restoring RB POP RB ; Always restore RB register . . . IRET ; RA = RAS, RAS = 0See also **PUSH RB RPTB #RSIZE. RC** RPTB #RSIZE, loc16



PUSH RB	B Push the RB Register onto the Stack										
Operands											
	RB		repeat block	register							
Opcode	LSW: 111	11 1111	1111 0000								
Description	Save the RB register on the stack. If a high-priority interrupt contains a RPTB instruction then the RB register must be stored on the stack before the RPTB block and restored after the RTPB block. In a low-priority interrupt RB must always be saved and restored. This save and restore must occur when interrupts are disabled.										
Flags	This instruction does not affect any flags floating-point Unit:										
	<b>Flag</b> Modified	TF No	<b>ZI</b> No	NI No	<b>ZF</b> No	NF No	LUF No	LVF No			
Pipeline	This is a	a single-c	cycle instruc	tion for the	first iteratio	n, and zero	o cycles the	eafter.			
Example	A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not have to save the RB register.										
	<pre>; Repeat Block within a High-Priority Interrupt (Non-Interruptible) _Interrupt: ; RAS = RA, RA = 0</pre>										
		 PUSH RB		; Save RB	register o	nly if a R	PTB block i	s used in the			
			ockEnd, AL	; Execute	the block i	AL+1 times					
	BlockEnd ; End of block to be repeated										
	 POP RB ; Restore RB register										
		::: IRET ; RA = RAS, RAS = 0									
	A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.										
	_Interr	upt:	within a Lo	w-Priority ; RAS = RA	-	(Interrupt	ible)				
	]	 PUSH RB 		; Always s	ave RB reg	ister					
	(	CLRC INTN	M	; Enable i	nterrupts	only after	saving RB				
		· · · · · · ·		; ISR may	or may not	include a	RPTB block	:			
	:	SETC INTN	M	; Disable	interrupts	before re	storing RB				
	]	POP RB		; Always r	restore RB :	register					
		 IRET		; RA = RAS	S, RAS = 0						
See also		3 RSIZE, F RSIZE, Id									





			pating-Point	v				
Operands								
	none		This instruct	ion does not	have any ope	rands		
Opcode	LSW: 111	.0 0101	0110 0010					
Description	The SA\ interrupt	Restore the floating-point register set (R0H - R7H and STF) from their shadow registers. The SAVE and RESTORE instructions should be used in high-priority interrupts. That is interrupts that cannot themselves be interrupted. In low-priority interrupt routines the floating-point registers should be pushed onto the stack.						
Restrictions	The RESTORE instruction cannot be used in any delay slots for pipelined operations. Doing so will yield invalid results. To avoid this, the proper number of NOPs or non-pipelined instructions must be inserted before the RESTORE operation.							
	MPYF3	<pre>; The following is INVALID     MPYF32 R2H, R1H, R0H ; 2 pipeline-cycle instruction (2p)     RESTORE ; INVALID, do not use RESTORE in a delay slot</pre>						ay slot
		2 R2H,	is VALID R1H, R0H					s instruction
Flags	Restorin	g the st	atus register	will overv	vrite all flags	8:		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	Yes	Yes	Yes	Yes	Yes	Yes	Yes



www.ti.com		Instruction						
Example	The following example shows a complete context save and restore for a high-priority interrupt. Note that the CPU automatically stores the following registers: ACC, P, XT, ST0, ST1, IER, DP, AR0, AR1 and PC. If an interrupt is low priority (that is it can be interrupted), then push the floating point registers onto the stack instead of using the SAVE and RESTORE operations.							
	; Interrupt Save _HighestPriorityISR: ASP PUSH RB PUSH AR1H:AR0H PUSH XAR2 PUSH XAR3 PUSH XAR4 PUSH XAR5 PUSH XAR6 PUSH XAR7 PUSH XT SPM 0 CLRC AMODE CLRC PAGE0,OVM SAVE RNDF32=1	<pre>; Uninterruptable ; Align stack ; Save RB register if used in the ISR ; Save other registers if used ; Save other registers ; Set default C28 modes ; Save all FPU registers ; set default FPU modes</pre>						
	; Interrupt Restore							
	RESTORE POP XT POP XAR7 POP XAR6 POP XAR5 POP XAR4 POP XAR3 POP XAR2 POP AR1H:AR0H POP RB	<pre>; Restore all FPU registers ; restore other registers ; restore RB register</pre>						
	NASP IRET	; un-align stack ; return from interrupt						

See also

SAVE FLAG, VALUE



RPTB label, loc16	PTB label, loc16 Repeat A Block of Code									
Operands										
	label		calculate RSI	ZE.	ssembler to det d immediately a			at block and to uded in the repeat		
	loc16		16-bit location	n for the repe	at count value.					
Opcode	LSW: 1011 MSW: 0000		0bbb bbbb loc16							
Description	Initialize re	epeat b	lock loop, re	peat coun	from [loc16	]				
Restrictions										
	The ma	aximum	n block size	is ≤127 16	bit words.					
	<ul> <li>An eve</li> </ul>	n align	ed block mu	st be $\geq$ 9 1	6-bit words.					
	<ul> <li>An odd</li> </ul>	l aligne	d block mus	t be ≥ 8 16	bit words.					
	<ul> <li>Interrup</li> </ul>	pts mus	st be disable	d when sa	ving or resto	oring the F	RB register.			
	Repeat	t blocks	s cannot be	nested.	-	-	-			
					ot allowed in terrupts are		beat block. T	his includes all		
	-		ecution ope		•	anoweu.				
Flags	This instruction does not affect any flags in the floating-point unit:									
	Flag T	ſF	ZI	NI	ZF	NF	LUF	LVF		
	Modified N	No	No	No	No	No	No	No		
Pipeline Example	<ul><li>This instruction takes four cycles on the first iteration and zero cycles thereafter. No special pipeline alignment is required.</li><li>The minimum size for the repeat block is 8 words if the block is even aligned and 9 words if the block is odd aligned. If you have a block of 8 words, as in the following example, you can make sure the block is odd aligned by proceeding it by a .align 2 directive and a NOP instruction. The .align 2 directive will make sure the NOP is even</li></ul>									
	aligned. Si	ince a l		-bit instruc				For blocks of		
	; Repeat Block of 8 Words (Interruptible)									
; ; find the largest element and put its addr .align 2						n XAR6				
	NOP RPTB MOVL			AR7 a	Execute th	e block A	R7+1 times			
	MOV32 MAXF32 MOVST0	R1H,* R0H,R NF,ZF	XAR0++ 1H		min size = max size =					
	MOVL VECTOR_MAX		ACC, LT		label indi RA is clea		end			
	copied to t automatica	the repo ally cop p was a	eat active sh ied back to	hadow (RA the RA bit.		the interi the hardv	upt exits, th vare to keep			
	A high pric	ority inte	errupt is defi	ined as an	interrupt tha	it cannot i	tself be inter	rupted. In a		

A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not



#### have to save the RB register.

```
; Repeat Block within a High-Priority Interrupt (Non-Interruptible)
;
; Interrupt:
                             ; RAS = RA, RA = 0
       PUSH RB
                             ; Save RB register only if a RPTB block is used in the
ISR
       . . .
        . . .
       RPTB #BlockEnd, AL ; Execute the block AL+1 times
       . . .
       . . .
       . . .
BlockEnd
                             ; End of block to be repeated
       . . .
        . . .
                             ; Restore RB register
       POP RB
       . . .
       IRET
                             ; RA = RAS, RAS = 0
```

A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.

; Repeat Block within a	Low-Priority Interrupt (Interruptible)
;	
; Interrupt:	; RAS = RA, RA = $0$
PUSH RB	; Always save RB register
CLRC INTM	; Enable interrupts only after saving RB
	; ISR may or may not include a RPTB block
SETC INTM	; Disable interrupts before restoring RB
POP RB	; Always restore RB register
IRET	; $RA = RAS$ , $RAS = 0$
POP RB	
PUSH RB	
RPTB label, RC	

See also



RPTB label, #RC	Repeat a Block of Code									
Operands										
	label		This label is used by the assembler to determine the end of the repeat block and to calculate RSIZE. This label should be placed immediately after the last instruction included in the repeat block.							
	#RC		16-bit immed	liate value for	the repeat cou	nt.				
Opcode	LSW: 1011 0101 1bbb bbbb MSW: cccc cccc cccc									
Description	Repeat a b	olock o	f code. The	repeat cou	int is specifie	ed as a im	mediate val	ue.		
Restrictions										
	The ma	aximun	n block size	is ≤127 16	-bit words.					
					16-bit words.					
		-	d block mus							
		•			aving or resto	orina the F	R register			
			s cannot be		aving of resid		CD register.			
	•				ot allowed ir	nside a rep	beat block. T	This includes all		
					terrupts are	allowed.				
			xecution ope							
Flags	This instruction does not affect any flags int the floating-point unit:									
	Flag T	F	ZI	NI	ZF	NF	LUF	LVF		
	Modified N	lo	No	No	No	No	No	No		
Pipeline	special pip	eline a	alignment is	required.	first iteration					
Example	words if th example, y directive a aligned. Si	e blocł /ou cai nd a N ince a	k is odd aligi n make sure OP instructi	hed. If you the block on. The .a bit instruc		k of 8 wor ed by proc ve will mal	ds, as in the eeding it by ke sure the l	following		
	_	Block (	Interruptik	ole)						
	; ; find the .align 2		est element	and put i	ts address i	n XAR6				
	NOP RPTB VECTOR_MAX_END, $\#(4-1)$ ; Execute the block 4 times MOVL ACC, XAR0 MOV32 R1H, $*XAR0++$ ; 8 or 9 words $\leq$ block size $\leq$ 127 word MAXF32 R0H, R1H MOVST0 NF, ZF									
								7 words		
	MOVL VECTOR_MAX		ACC, LT		; RE indicat ; RA is clea		d address			
	copied to t automatica	he rep ally cop p was	eat active sl bied back to	hadow (RA the RA bit	ctive (RA) bi .S) bit. Wher . This allows errupt is take	the interi the hardv	rupt exits, th vare to keep	track if a		
			errupt is def	ined as an	interrupt tha	at cannot i	tself be inte	rrupted. In a		

A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not



#### have to save the RB register.

```
; Repeat Block within a High-Priority Interrupt (Non-Interruptible)
;
; Interrupt:
                             ; RAS = RA, RA = 0
       PUSH RB
                             ; Save RB register only if a RPTB block is used in the
ISR
       . . .
        . . .
       RPTB #BlockEnd, #5 ; Execute the block 5+1 times
       . . .
       . . .
       . . .
BlockEnd
                             ; End of block to be repeated
       . . .
        . . .
                             ; Restore RB register
       POP RB
       . . .
       IRET
                             ; RA = RAS, RAS = 0
```

A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.

; Repeat Block within a	Low-Priority Interrupt (Interruptible)
i	
; Interrupt:	; $RAS = RA$ , $RA = 0$
PUSH RB	; Always save RB register
CLRC INTM	; Enable interrupts only after saving RB
• • •	; ISR may or may not include a RPTB block
•••	
•••	
SETC INTM	; Disable interrupts before restoring RB
•••	
POP RB	; Always restore RB register
• • •	
IRET	; $RA = RAS$ , $RAS = 0$
POP RB	
PUSH RB	

See also

**RPTB #RSIZE**, loc16



### SAVE FLAG, VALUE Save Register Set to Shadow Registers and Execute SETFLG

Operands								
	FLAG		11 bit mask i	ndicating wh	ch floating-poi	nt status flags	to change.	
	VALUE		11 bit mask i	ndicating the	flag value; 0 c	or 1.		
Opcode		10 0110 FF FVVV	01FF FFFF VVVV VVVV					
Description	STF) to a single are chai execute interrup	the shad cycle. T nged. Th d. The S ts. That i		set and co gister is co VM] flag is STORE ir hat canno	mbines the pied to the set to 1 wh structions s t themselves	SETFLG F shadow reg en the SA hould be u s be interru	LAG, VALU gister before VE comman sed in high- ipted. In low	E operation in the flag values d has been priority -priority
Restrictions	Do not use the SAVE instruction in the delay slots for pipelined operations. Doing so of yield invalid results. To avoid this, the proper number of NOPs or non-pipelined instructions must be inserted before the SAVE operation. ; The following is INVALID MPYF32 R2H, R1H, R0H ; 2 pipeline-cycle instruction (2p) SAVE RNDF32=1 ; INVALID, do not use SAVE in a delay slot ; The following is VALID MPYF32 R2H, R1H, R0H ; 2 pipeline-cycle instruction (2p) NOP ; 1 delay cycle, R2H updated after this instruction							elined
	SAVE	RNDF32	=1	; VALID		_		
Flags	This ins	truction I	modifies the	following f	lags in the S	STF registe	r:	
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Pipeline Example	This is a To make the STF SAVE MOVS	a single-o e it easie LG oper <sup>RNDF</sup> T <sup>0</sup> TF, owing ex t. Note th	ation as show 32=0, TF=1, ZF, LUF ample shows nat the CPU a	ion. egible, the wn below: <sup>ZF=0</sup> ; F ; C ; N ; T s a comple automatica	A assembler LAG = 01001 opy the ind ote: X mean he assemble ete context s ally stores th	000100, VA icated fla s this fla r will set cave and re	LUE = X0XX0 gs to ST0 g will not these X va estore for a h	be modified. llues to 0. high priority
			DP, AR0, AR					



Instructions

_HighestPriorityISR:	
ASP	; Align stack
PUSH RB	; Save RB register if used in the ISR
PUSH AR1H:AR0H	; Save other registers if used
PUSH XAR2	
PUSH XAR3	
PUSH XAR4	
PUSH XAR5	
PUSH XAR6	
PUSH XAR7	
PUSH XT	
SPM 0	; Set default C28 modes
CLRC AMODE	
CLRC PAGE0,OVM	
SAVE RNDF32=0	; Save all FPU registers
	; set default FPU modes
• • •	
RESTORE	; Restore all FPU registers
POP XT	; restore other registers
POP XAR7	
POP XAR6	
POP XAR5	
POP XAR4	
POP XAR3	
POP XAR2	
POP AR1H:AR0H	
POP RB	; restore RB register
NASP	; un-align stack
IRET	; return from interrupt
RESTORE	
RESIDRE	F

See also

SETFLG FLAG, VALUE



### SETFLG FLAG, VALUE Set or clear selected floating-point status flags

Operands							
	FLAG	11 bit mask i	ndicating which floating	ng-point statu	us flags to cha	inge.	
	VALUE	11 bit mask i	ndicating the flag valu	ue; 0 or 1.			
Opcode	LSW: 1110 MSW: FFFF						
Description	STF registe changed. T flags will no	LG instruction is user. The FLAG field That is, if a FLAG f ot be modified. Th	is an 11-bit valu bit is set to 1 it ind e bit mapping of t	e that indio dicates tha the FLAG	cates which at flag will b field is show	flags will b e changed;	all other
10 9	8 7		5 4	3	2	1	0
reserved RNDF32	reserved rese	I I	ZI NI	ZF	NF	LUF	LVF
	The VALU	E field indicates th	e value the flag s	should be s	set to; 0 or	1.	
Restrictions	can yield ir	the SETFLG instruction of the SETFLG instruction of the second se	woid this, the pro	per numbe	er of NOPs		
	; The following is INVALID MPYF32 R2H, R1H, R0H ; 2 pipeline-cycle instruction (2p) SETFLG RNDF32=1 ; INVALID, do not use SETFLG in a delay slot						
Flags	MPYF32 NOP SETFLG	owing is VALID R2H, R1H, R0H RNDF32=1	; 2 pipeline- ; 1 delay cyc ; VALID	le, R2H up	dated afte		truction
Flags	MPYF32 NOP SETFLG This instruc	R2H, R1H, R0H RNDF32=1 ction modifies the	; 1 delay cycl ; VALID following flags in	the STF re	egister:	r this ins	
Flags	MPYF32 NOP SETFLG This instruct Flag T	R2H, R1H, R0H RNDF32=1 Ction modifies the F ZI	; 1 delay cycl ; VALID following flags in NI ZF	the STF re	egister:	r this ins	.VF
Flags	MPYF32 NOP SETFLG This instruct Flag T	R2H, R1H, R0H RNDF32=1 ction modifies the	; 1 delay cycl ; VALID following flags in	the STF re	egister:	r this ins	
Flags	MPYF32 NOP SETFLG This instruct Flag T Modified Y	R2H, R1H, R0H RNDF32=1 Ction modifies the F ZI	; 1 delay cycl ; VALID following flags in <u>NI ZF</u> Yes Yes	the STF re	egister:	r this ins	.VF
Flags Pipeline	MPYF32 NOP SETFLG This instruct Flag T Modified Y Any flag ca	R2H, R1H, R0H RNDF32=1 Ction modifies the F ZI es Yes	; 1 delay cycl ; VALID following flags in <u>NI ZF</u> Yes Yes this instruction.	the STF re	egister:	r this ins	.VF
-	MPYF32 NOP SETFLG This instruct Flag Ti Modified Y Any flag ca This is a si To make it	R2H, R1H, R0H RNDF32=1 ction modifies the F ZI es Yes	i 1 delay cycl i VALID following flags in NI ZF Yes Yes this instruction. ion.	the STF re	edated afte egister: = L es Y	r this ins UF I es N	_VF (es
Pipeline	MPYF32 NOP SETFLG This instruct Flag Ti Modified Y Any flag ca This is a si To make it	R2H, R1H, R0H         RNDF32=1         ction modifies the         F       ZI         es       Yes         an be modified by angle-cycle instruct         easier and legible	; 1 delay cycl ; VALID following flags in <u>NI ZF</u> Yes Yes this instruction. ion. e, the assembler wo below: ., ZF=0 ; FI ; Cd ; X	the STF re the STF re Ye will accept LAG = 0100 opy the in means thi	edated afte egister: = L es Y	r this ins UF I es N ALUE synta ALUE = X0X ags to ST0 not modifi	_VF (es ax for the x1xx0xxx ed.



## SUBF32 RaH, RbH, RcH 32-bit Floating-Point Subtraction

#### Operands

Operands											
	RaH	f	loating-	point destination i	egister (R0H	to R1)					
	RbH	1	loating-	point source regis	ter (R0H to R	R1)					
	RcH	1	loating-	point source regis	ter (R0H to R	81)					
<b>.</b> .											
Opcode	LSW: 1110 0111 0010 0000 MSW: 0000 000c ccbb baaa										
Description	Subtract the contents of two floating-point registers RaH = RbH - RcH										
Flags	This inst	ruction mo	difies	the following f	ags in the	STF registe	r:				
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No	Yes	Yes			
	The STF	- register fl	ags ar	e modified as	follows:						
	• LUF	= 1 if MPY	F32 ge	enerates an u	nderflow co						
	<ul> <li>LVF</li> </ul>	= 1 if MPY	F32 ge	enerates an ov	verflow con	dition.					
Pipeline	This is a 2 pipeline cycle (2p) instruction. That is:										
	SUBF32 RaH, RbH, RcH ; 2 pipeline cycles (2p) NOP ; 1 cycle delay or non-conflicting instruction										
	; < SUBF32 completes, RaH updated NOP										
	Any instruction in the delay slot must not use RaH as a destination register or as a source operand.										
Example	Calculat	e Y - A + E	3 - C:								
	MOVL MOV32 MOVL	XAR4, #2 ROH, *2 XAR4, #1	KAR4	; Load ROH w	ith A						
	MOV32 MOVL	R1H, *2 XAR4, #0	KAR4 C	; Load R1H w	ith B						
		2 ROH,R1H		; Add A + B	-	allel					
	MOV32	R2H,*XA	₹4	; Load R2H w ; < MOV32							
	MOVL	XAR4,#	ĸt	· 300E22	aomplete						
	SUBF3 NOP	2 ROH,ROH	,R2H	; < ADDF32 ; Subtract C	-	в)					
	MOV32	*XAR4,R	ЭH	; < SUBF32 ; Store the	-						
See also	SUBF32	RaH, #16	FHi, R	bH							
	SUBF32	RdH, ReF	I, RfH	MOV32 Ral							
	SUBF32	: RdH, ReF	i, RfH	MOV32 mei	m32, RaH						

SUBF32 RdH, ReH, RfH || MOV32 mem32, RaH MPYF32 RaH, RbH, RcH || SUBF32 RdH, ReH, RfH

### SUBF32 RaH, #16FHi, RbH 32-bit Floating Point Subtraction

Operands										
	RaH	floating-po	int destination	register (R0H	to R1)					
	#16FHi			e that represent low 16-bits of t						
	RbH	floating-po	int source reg	ister (R0H to R	1)					
Opcode		LSW: 1110 1000 11II IIII MSW: IIII IIII IIbb baaa								
Description	Subtract RbH the result of th			alue represe	ented by the	e immediate	operand. Store			
	#16FHi is a 16 floating-point w most useful fo Some example (0xBFC00000 That is, the va RaH = #16FHi:	value. The lov r representin es are 2.0 (0) . The assem lue -1.5 can	w 16-bits of g constants x40000000 bler will ac	f the mantise where the ), 4.0 (0x408 cept either a	sa are assur lowest 16-b 300000), 0.5 a hex or floa	med to be a its of the ma 5 (0x3F0000 it as the imn	ll 0. #16FHi is antissa are 0. 000), and -1.5			
Flags	This instructio	n modifies th	e following	flags in the	STF registe	er:				
	Flag TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified No	No	No	No	No	Yes	Yes			
Pipeline	<ul> <li>LVF = 1 if</li> <li>This is a 2 pip</li> </ul>	MPYF32 gen MPYF32 gen eline cycle (2 ан, #16FHi,	erates an c 2p) instructi RbH ; 2 pi ; 1 cy	overflow con on. That is:	dition. es (2p) r non-confl		ruction			
	Any instruction source operar		slot must i	not use RaH	l as a destir	nation regist	er or as a			
Example	Calculate Y =		:							
	MOV32 R0H MOVL XAR MOV32 R1H			with B	allel					
	SUBF32 R0H NOP	,#2.0,R2H ;		(A + B) fro	m 2.0					
	MOV32 *XA		Store the	2 completes result						
See also	SUBF32 RaH, SUBF32 RdH, SUBF32 RdH, MPYF32 RaH	ReH, RfH    ReH, RfH	MOV32 me	em32, RaH	RfH					



### SUBF32 RdH, ReH, RfH MOV32 RaH, mem32 32-bit Floating-Point Subtraction with Parallel Move

Operands										
	RdH floating-point destination register (R0H to R7H) for the SUBF32 operation RdH cannot be the same register as RaH									
	ReH	floating-point source register (R0H to R7H) for the SUBF32 operation								
	RfH		floating-point	source register	r (R0H to R7H)	for the SUBF	32 operation			
	RaH			destination reg	ister (R0H to F gister as RdH	R7H) for the M	OV32 operati	on		
	mem32		pointer to 32-	bit source men	nory location fo	r the MOV32 of	operation			
Opcode	LSW: 111 MSW: eed		0010 fffe mem32							
Description	Subtract the contents of two floating-point registers and move from memory to a floating-point register. RdH = ReH - RfH, RaH = [mem32]									
Restrictions			egister for th me register a		and the MO	V32 must b	e unique. T	Гhat is, RaH		
Flags	This instruction modifies the following flags in the STF register:									
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified	No	Yes	Yes	Yes	Yes	Yes	Yes		
		-	flags are mo BF32 genera			tion.				
	• LVF =	= 1 if SU	BF32 genera	ates an ovei	flow conditi	on.				
	The MO	/32 Instr	uction will se	et the NF. Z	F. NI and ZI	flags as fo	llows:			
	<pre>The MOV32 Instruction will set the NF, ZF, NI and ZI flags as follows: NF = RaH(31); ZF = 0; if(RaH(30:23) == 0) { ZF = 1; NF = 0; } NI = RaH(31); ZI = 0; if(RaH(31:0) == 0) ZI = 1;</pre>									
Pipeline			peline-cycle	instruction (	(2n) and MC	)\/32 takas	a sinala cv	rcla. That is:		
ripenne		32 RdH,	ReH, RfH mem32	; 2 pipeli ; 1 cycle ; < MOV3 ; 1 cycle	ne cycles ( 2 completes delay or nc	2p) , RaH updat m-conflict:	ted ing instru			
	NOP			; < SUBF	32 complete	s, RdH upda	ated			
Any instruction in the delay slot must not use RdH as a destination regi							on reaister	r or as a		

Any instruction in the delay slot must not use RdH as a destination register or as a source operand.



1.000		
Instri	ictions	

Example	<pre>MOVL XAR1, #0xC000 ; XAR1 = 0xC000 SUBF32 R0H, R1H, R2H ; (A) R0H = R1H - R2H    MOV32 R3H, *XAR1 ;</pre>
	ADDF32 R5H, R4H, R3H ; (B) R5H = R4H + R3H    MOV32 *+XAR1[4], R0H ; ; < R0H stored MOVL XAR2, #0xE000 ;
	; < (B) completes, R5H valid MOV32 *XAR2, R5H ; ; < R5H stored
See also	SUBF32 RaH, RbH, RcH SUBF32 RaH, #16FHi, RbH

MPYF32 RaH, RbH, RcH || SUBF32 RdH, ReH, RfH



### SUBF32 RdH, ReH, RfH MOV32 mem32, RaH 32-bit Floating-Point Subtraction with Parallel Move

Operands										
epolaliao	RdH	floating-point	t destination r	aister (ROH +c	R7H) for the	SUBE32 oper	ation			
	ReH	floating-point destination register (R0H to R7H) for the SUBF32 operation floating-point source register (R0H to R7H) for the SUBF32 operation								
	RfH	floating-point source register (R0H to R7H) for the SUBF32 operation								
	mem32	• •	•	n memory loca	,	•				
	RaH			er (R0H to R7		•				
					,	•				
Opcode	LSW: 1110 000 MSW: eedd daa									
Description	Subtract the contents of two floating-point registers and move from a floating-point register to memory.									
	RdH = ReH - 1 [mem32] = Ral									
Flags	This instruction	on modifies the H, mem32	following fla	ags in the S	TF register	:SUBF32 R	dH, ReH, RfH			
	Flag TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified No	No	No	No	No	Yes	Yes			
Pipeline	• LVF = 1 if	SUBF32 gener SUBF32 gener 2 pipeline-cycle	rates an ov	erflow cond	ition.	s a single c	cycle. That is:			
	SUBF32	RdH, ReH, RfH nem32, RaH	, RfH ; 2 pipeline cycles (2p)							
	NOP			-	mpletes, mem32 updated y or non-conflicting instruction					
	; < ADDF32 completes, RdH updated									
	Any instruction source opera	n in the delay s	slot must no	ot use RdH :	as a destina	ation registe	er or as a			
Example	ADDF32	R3H, R6H, R4 R7H, *-SP[2		; (A) R31 ;	H = R6H + F	AH and R7H	I = I3			
	GUDE20		4 TT	; < R71		411				
	SUBF32	R6H, R6H, R4	4H		H = R6H - F DF32 (A) cc		.3H valid			
	SUBF32	R3H, R1H, R' *+XAR5[2], I		; (C) R31 :	H = R1H - F	R7H and sto	re R3H (A)			
	110132	TARCEZ,		; < SU	BF32 (B) cc	ompletes, R	6H valid			
	ADDF32	R4H, R7H, R1	14		V32 complet D = R7H + F					
	MOV32	*+XAR5[6], I		;						
					BF32 (C) co V32 complet	-	-			
	MOV32	*+XAR5[0], H	R3H	; store ]	-					
					DF32 (D) cc	-				
	MOV32	*+XAR5[4], I	R4H	; store ] ; < MO	R4H (D) V32 complet	es, (D) st	ored			
See also	SUBF32 RaH					,				
000 aigu		I, #16FHi, RbH								
	SUBF32 RdF	I, ReH, RfH    N								
	MPYF32 RaH	I, RbH, RcH    S	SUBF32 Ro	IH, ReH, Rf	Н					



### SWAPF RaH, RbH{, CNDF} Conditional Swap

#### Operands

erands							
	RaH	• •	oint register (R	,			
	RbH	• •	oint register (R	0H to R7H)			
	CNDF	condition	tested				
code	LSW: 1110 MSW: 0000						
scription	Conditional	swap of RaH a	and RbH.				
		= true) swap R					
	CNDF is or	ne of the follow	na conditio	ns:			
	Encode <sup>(1)</sup>	CNDF	-	ription		STF Flags Teste	d
	0000	NEQ	Not e	equal to zero		ZF == 0	
	0001	EQ	Equa	l to zero		ZF == 1	
	0010	GT	Grea	ter than zero		ZF == 0 AND NF	== 0
	0011	GEQ	Grea	ter than or equ	al to zero	NF == 0	
	0100	LT	Less	than zero		NF == 1	
	0101	LEQ	Less	than or equal t	o zero	ZF == 1 AND NF	== 1
	1010	TF	Test	flag set		TF == 1	
	1011	NTF	Test	flag not set		TF == 0	
	1100	LU	Latch	ed underflow		LUF == 1	
	1101	LV	Latch	ed overflow		LVF == 1	
	1110	UNC	Unco	nditional		None	
	1111	UNCF <sup>(2)</sup>		nditional with f	lag	None	
js	(2) This is the condition a condition modify the	ot shown are reserved default operation will allow the ZF, I onal operation is expesse flags. Stion modifies the E ZI	if no CNDF find NF, ZI, and NI Accuted. All oth	flags to be mod er conditions v	dified wher vill not		LVF
	Modified N	o No	No	No	No	No	No
peline	No flags af						
mple	;find the	largest elemen	t and put i	t in R1H			
	MOVL MOV32 .align	XAR1, #0xB R1H, *XAR1 2		itialize Rl	Н		
	NOP RPTB MOV32 CMPF32 SWAPF NOP NOP	LOOP_END, R2H, *XAR1 R2H, R1H R1H, R2H,	++ ; Up ; Cc GT ; Sw ; Fc	ecute the k date R2H wi mpare R2H w ap R1H and r minimum r r minimum r	th next vith R1H R2H if F epeat bl	element 2 > R1 .ock size	

See also

LOOP\_END:



#### **TESTTF CNDF** Test STF Register Flag Condition

#### Operands

CNDF condition to test

Opcode

Description

LSW: 1110 0101 1000 CNDF

Test the floating-point condition and if true, set the TF flag. If the condition is false, clear the TF flag. This is useful for temporarily storing a condition for later use.

if (CNDF == true) TF = 1; else TF = 0;

#### CNDF is one of the following conditions:

Encode <sup>(1)</sup>	CNDF	Description	STF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 AND NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF <sup>(2)</sup>	Unconditional with flag modification	None

(1) Values not shown are reserved.

(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF, NF, ZI, and NI flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

#### Flags

This instruction modifies the following flags in the STF register:

Flag	TF	ZI	NI	ZF	NF	LUF	LVF	
Modified	Yes	No	No	No	No	No	No	

TF = 0; if (CNDF == true) TF = 1;

ROH, ROH

This is a single-cycle instruction.

NEGF32

End

Note: If (CNDF == UNC or UNCF), the TF flag will be set to 1.

**Pipeline** 

Example

CMPF32 ROH, #0.0 ; Compare ROH against 0 ; Set TF if ROH less than 0 (NF == 0) TESTTF LTABS ROH, ROH ; Get the absolute value of ROH ; Perform calculations based on ABS ROH ; Copy TF to TC in STO MOVST0 TF End, NTC SBF ; Branch to end if TF was not set

### See also



### UI16TOF32 RaH, mem16 Convert unsigned 16-bit integer to 32-bit floating-point value

Operands									
	RaH	floating-point	destinatior	register (R0H	to R7H)				
	mem16	pointer to 16-	bit source	memory locatio	on				
		•		,					
Opcode	LSW: 1110 0	010 1100 0100							
	MSW: 0000 0	aaa mem16							
Description		erting F32 to I16/U ne F32TOI16R/UI							
	RaH = UI16T	oF32[mem16]							
Flags	This instruct	ion does not affe	ct any fla	igs:					
	Flag TF	ZI	NI	ZF	NF	LUF	LVF		
	Modified No	No	No	No	No	No	No		
Pipeline	This is a 2 p	pipeline cycle (2p)	instructi	on. That is:					
		32 RaH, meml6		pipeline cy					
	NOP					flicting in			
	; < UI16TOF32 completes, RaH updated NOP								
	Any instruct	ion in the delay of		aat ugo Dol	l oo o dootir	otion regist			
	source oper	ion in the delay sl	iot must i	lot use Kar		lation registi			
	source oper	anu.							
Example	; float32 y								
		ESULT0 is an uns : y = (float)Adc			+ h:				
	;	• y = (110ac)Auc	Regs.ADC		1 07				
	MOVW	DP @0x01C4							
		2 ROH, @8			AdcRegs.RES	SULTO			
	MOV32	R1H, *-SP[6]		H = M - Conversio	on complete,	ROH walid			
	MPYF32	ROH, R1H, ROH		H = (float)		Rom Varia			
	MOV32	R1H, *-SP[8]		H = B					
					mplete, ROH				
	ADDF32 NOP	ROH, ROH, RIH	; R0	H = Y = (f)	.oat)X * M +	- В			
					mplete, ROH	I valid			
	MOV32	*-[SP], ROH	; St	ore Y					
See also	F32TOI16 R	aH, RbH							
	F32TOI16R	RaH, RbH							
	F32TOUI16	RaH, RbH							
	F32TOUI16	R RaH, RbH							
	116TOF32 R	RaH, RbH							
	116TOF32 R	RaH, mem16							
	UI16TOF32	RaH, RbH							



### UI16TOF32 RaH, RbH Convert unsigned 16-bit integer to 32-bit floating-point value

Operatios											
	RaH		floating-point	destinatior	n register (R0H	to R7H)					
	RbH		floating-point	source reg	gister (R0H to F	R7H)					
									_		
Opcode	LSW: 111	LO 0110	1000 1111								
	MSW: 000	0000 000	00bb baaa								
Description			ng F32 to I16/L F32TOI16R/UI						0		
	RaH = UI	[16ToF3	2[RbH]								
Flags	This inst	truction	does not affe	ct any fla	ags:						
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No	No	No			
Pipeline	This is a	This is a 2 pipeline cycle (2p) instruction. That is:									
	UI16 NOP	5TOF32	RaH, RbH	; 1		or non-con	nflicting ir , RaH update				
	NOP					-	-				
	Any inst source o		in the delay sl d.	lot must	not use Ra⊦	l as a desti	nation regist	er or as a			
Example	MOVX		R5H, #0x800F		15:0] = 327						
	UI16 NOP	5TOF32	R6H, R5H	; 1 cy	= UI16TOF32 rcle delay f = 32783.0 (	or UI16TOF	32 to comple	ete			
See also	F32TOI										
	F32TOI F32TOL										
			RaH, RbH								
	I16TOF										
			l, mem16								
	01010	UI16TOF32 RaH, mem16									

### UI32TOF32 RaH, mem32 Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value

Operando											
	RaH	0.1		n register (R0H	,						
	mem32	pointer to 32	2-bit source	memory locati	on						
Opcode	LSW: 1110 00 MSW: 0000 02	010 1000 0100 aaa mem32									
Description	RaH = UI32To	oF32[mem32]									
Flags	This instruct	This instruction does not affect any flags:									
	Flag TF	ZI	NI	ZF	NF	LUF	LVF				
	Modified No	No	No	No	No	No	No				
Pipeline	This is a 2 p	ipeline cycle (2p	) instruct	ion. That is:							
	UI32TOF NOP NOP	32 RaH, mem32	; 1		non-conf	licting inst , RaH update					
	Any instructi source oper	on in the delay and.	slot must	not use Ral	H as a destir	nation regist	er or as a				
Example			M + B								
	; UI32TOF3:	2 ROH, *-SP[2]	; R0	H = (float	) X						
	MOV32	R1H, *-SP[6]		.H = M							
	MPYF32	ROH, R1H, RO		H = (float	on complete )X * M	, RUH VAIIU					
	MOV32	R1H, *-SP[8]		.H = B - MDVF32 c	omplete, ROB	I valid					
	ADDF32 NOP	ROH, ROH, R1			loat)X * M -						
	MOV32	*-[SP], ROH		- ADDF32 co core Y	omplete, ROB	H valid					
See also	F32TOI32 R F32TOUI32 I32TOF32 R I32TOF32 R UI32TOF32	RaH, RbH aH, mem32 aH, RbH									



### UI32TOF32 RaH, RbH Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value

Operanus											
	RaH	floating-point	destination	register (R0H	to R7H)			_			
	RbH	floating-point :	source regis	ster (R0H to F	R7H)						
Opcode	LSW: 1110 0110	1000 1011									
	MSW: 0000 0000	00bb baaa									
Description	RaH = UI32ToF3	2 RbH									
Flags	This instruction	This instruction does not affect any flags:									
	Flag TF	ZI	NI	ZF	NF	LUF	LVF				
	Modified No	No	No	No	No	No	No				
	This is a Queins			. Thetie							
Pipeline	This is a 2 pipeline cycle (2p) instruction. That is:										
	UI32TOF32 NOP	UI32TOF32 RaH, RbH ; 2 pipeline cycles (2p) NOP ; 1 cycle delay or non-conflicting instruction									
	NOF					, RaH update					
	NOP										
	Any instruction		ot must n	ot use RaH	l as a desti	nation registe	er or as a				
	source operand	d.									
Example	MOVIZ	R3H, #0x8000	-	-							
	MOVXI	R3H, #0x1111	-	5:0] = 0x 214748801							
	UI32TOF32	R4H, R3H	; R4H =	UI32TOF32	2 (R3H)						
	NOP		-	-	or UI32TOF: .7.0 (0x4F0)	32 to comple 00011)	te				
See also	F32TOI32 RaH	I. RbH									
	F32TOUI32 Ra										
	I32TOF32 RaH	l, mem32									
	I32TOF32 RaH										
	UI32TOF32 Ra	aH, mem32									



ZERO RaH	Zero the	Float	ing-Point Re	gister Ra	aH			
Operands								
	RaH		floating-point	t register (R	ROH to R7H)			
Opcode			T.SM • 111	0 0101	1001 0aa	2		
						a		
Description	Zero the	indicat	ted floating-po	oint regist	ter:			
			RaH = 0					
Flags	This inst	ruction	modifies the	following	flags in the	STF register		
	Flag	TF	ZI	NI	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No	No	No
_xampio		+= (x	[2*i] * y[2*:		-			
	;{ ; real	+= (x += (x	[2*i] * y[2*: [2*i] * y[2*:		-			
	;{ ; real ; imag ;}	+= (x += (x AR7 = :	[2*i] * y[2*: [2*i] * y[2*: n-1 H	i+1]) + ( ; F	-	y[2*i]); 0		
	;{ ; real ; imag ;} ;Assume ZERO ZERO LOOP MOV MOV	+= (x += (x AR7 = : R4: R5: AL AC	[2*i] * y[2*: [2*i] * y[2*: n-1 H H , AR7 C, AL << 2	i+1]) + ( ; F	x[2*i+1] *	y[2*i]); 0		
p.o	;{ ; real ; imag ;} ;Assume ZERO LOOP MOV MOV MOV MOV MOV3 MOV3	+= (x += (x AR7 = : R4: R5: AL AC AR 2 R0: 2 R1:	[2*i] * y[2*: [2*i] * y[2*: n-1 H H , AR7 C, AL << 2 0, ACC H, *+XAR4[AR( H, *+XAR5[AR(	i+1]) + ( ; r ; r ) ; r	x[2*i+1] *	y[2*i]); 0 0		
	;{ ; real ; imag ;} ;Assume ZERO ZERO LOOP MOV MOV MOV MOV MOV MOV MOV MOV MOV MOV	+= (x += (x AR7 = : R4: R5: AL AC AR 2 R0: 2 R1: AR 32 R6: 2 R2:	[2*i] * y[2*: [2*i] * y[2*: n-1 H H H C, AL << 2 0, ACC H, *+XAR4[AR( H, *+XAR5[AR( 0, #2 H, R0H, R1H; H, *+XAR4[AR(	i+1]) + ( ; r ; r ; r ; r 0] ; r 0] ; r 0] ; r	x[2*i+1] * 24H = real = 25H = imag = 20H = x[2*i 21H = y[2*i 26H = x[2*i 22H = x[2*i	<pre>y[2*i]); 0 0 1 ] ] * y[2*i] +1]</pre>		
	;{ ; real ; imag ;} ;Assume ZERO ZERO LOOP MOV MOV MOV MOV MOV3 MOV3 ADD MPYF    MOV3 MPYF    MOV3	+= (x += (x AR7 = : R4: R5: AL AC AR 2 R0: 2 R1: AR 32 R6: 2 R2: 32 R1: 2 R3:	[2*i] * y[2*: [2*i] * y[2*: n-1 H H H H C, AL << 2 0, ACC H, *+XAR4[AR( H, *+XAR5[AR( 0, #2 H, R0H, R1H; H, *+XAR4[AR( H, R1H, R2H H, *+XAR5[AR(	i+1]) + ( ; r ; r )] ; r )] ; r )] ; r )] ; r )] ; r )] ; r	x[2*i+1] * 24H = real = 25H = imag = 26H = x[2*i 21H = y[2*i 26H = x[2*i 22H = x[2*i 22H = x[2*i 23H = y[2*i 23H = y[2*i	<pre>y[2*i]); 0 0 1 ] ] * y[2*i] +1] ] * x[2*i+2 +1]</pre>		
	;{ ; real ; imag ;} ;Assume ZERO ZERO LOOP MOV MOV MOV MOV MOV MOV3 MOV3 MOV3 MOV3	+= (x += (x AR7 = : R4: R5: AL AC AR 2 R0: 2 R1: AR 32 R6: 2 R2: 32 R1: 2 R3: 32 R2: 32 R2: 32 R4:	[2*i] * y[2*: [2*i] * y[2*: n-1 H H H C, AL << 2 0, ACC H, *+XAR4[AR( H, *+XAR5[AR( 0, #2 H, R0H, R1H; H, R1H, R2H	i+1]) + ( ; r ; r )] ; r )] ; r )] ; r ; r )] ; r ; r ; r ; r	x[2*i+1] * 24H = real = 25H = imag = 25H = imag = 25H = 100000000000000000000000000000000000	<pre>y[2*i]); 0 0 0 ] ] 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 *</pre>	+1]	
Example	;{ ; real ; imag ;} ;Assume ZERO ZERO LOOP MOV MOV MOV MOV MOV3 MOV3 ADD MPYF    MOV3 MPYF    MOV3 MPYF    MOV3	+= (x += (x AR7 = : R4: R5: AL AC AR 2 R0: 2 R1: AR 32 R6: 32 R1: 32 R2: 32 R1: 32 R2: 32 R4: 32 R0: 32 R5: 32 R4: 32 R5:	<pre>[2*i] * y[2*: [2*i] * y[2*: n-1 H H H H C, AL &lt;&lt; 2 0, ACC H, *+XAR4[AR( H, *+XAR5[AR( 0, #2 H, R0H, R1H; H, *+XAR4[AR( H, R1H, R2H H, *+XAR5[AR( H, R2H, R3H H, R4H, R6H</pre>	i+1]) + ( ; r ; r ; r ) ; r ) ; r ) ; r ) ; r ; r ) ; r ; r ; r ; r ; r ; r ; r ; r ; r ; r	x[2*i+1] * 24H = real = 25H = imag = 25H = imag = 25H = imag = 25H = 12*i = 22*i	<pre>y[2*i]); 0 0 0 ] ] * y[2*i] +1] * x[2*i+2 +1] * y[2*i] ] * y[2*i] ] * y[2*i] ] * y[2*i+1] ] * y[2*i+2 +1] * y[2*i+2 +1] * y[2*i+2 +1] * y[2*i+2]</pre>	+1] ] ; +1]	



```
ZEROA
                       Zero All Floating-Point Registers
Operands
                        none
Opcode
                       LSW: 1110 0101 0110 0011
                       Zero all floating-point registers:
Description
                       ROH = 0
                       R1H = 0
                       R2H = 0
                       R3H = 0
                       R4H = 0
                       R5H = 0
                       R6H = 0
                       R7H = 0
                       This instruction modifies the following flags in the STF register:
Flags
                       Flag
                                TF
                                         ΖI
                                                     NI
                                                                ZF
                                                                           NF
                                                                                      LUF
                                                                                                 LVF
                       Modified
                                No
                                         No
                                                     No
                                                                No
                                                                           No
                                                                                      No
                                                                                                 No
                       No flags affected.
Pipeline
                       This is a single-cycle instruction.
Example
                       ifor(i = 0; i < n; i++)
                       ;{
                           real += (x[2*i] * y[2*i]) - (x[2*i+1] * y[2*i+1]);
                       ;
                           imag += (x[2*i] * y[2*i+1]) + (x[2*i+1] * y[2*i]);
                       ;
                       ;Assume AR7 = n-1
                           ZEROA
                                                        ; Clear all RaH registers
                       LOOP
                           MOV
                                   AL, AR7
                           MOV
                                   ACC, AL << 2
                           MOV
                                   AR0, ACC
                           MOV32
                                   ROH, *+XAR4[AR0]
                                                        ; R0H = x[2*i]
                                   R1H, *+XAR5[AR0]
                           MOV32
                                                        ; R1H = y[2*i]
                                   AR0,#2
                           ADD
                           MPYF32 R6H, R0H, R1H;
                                                        ; R6H = x[2*i] * y[2*i]
                                   R2H, *+XAR4[AR0]
                                                        ; R2H = x[2*i+1]
                        || MOV32
                           MPYF32 R1H, R1H, R2H
                                                       ; R1H = y[2*i] * x[2*i+2]
                           MOV32 R3H, *+XAR5[AR0]
MPYF32 R2H, R2H, R3H
                        || MOV32
                                                       ; R3H = y[2*i+1]
                                                       ; R2H = x[2*i+1] * y[2*i+1]
                        | ADDF32 R4H, R4H, R6H
                                                       ; R4H += x[2*i] * y[2*i]
                           MPYF32 ROH, ROH, R3H
                                                       ; ROH = x[2*i] * y[2*i+1]
                        ADDF32
                                   R5H, R5H, R1H
                                                        ; R5H += y[2*i] * x[2*i+2]
                                                       ; R4H -= x[2*i+1] * y[2*i+1]
                           SUBF32 R4H, R4H, R2H
                                                       ; R5H += x[2*i] * y[2*i+1]
                           ADDF32 R5H, R5H,R0H
                                   LOOP , AR7--
                           BANZ
                       ZERO RaH
See also
```

Instructions



# **Revision History**

### A.1 Changes

This revision history lists the technical changes made in the most recent revision.

Location	Additions, Deletions, Modifications
Figure 1-1	Modified the functional block diagram
Section 1.2.1	Added this section.
Section 1.3.1	Deleted part of the last bullet in Emulation Logic section
Section 1.4.1	Modified bullets in Address and Data Buses section
Example 2-2	Modified code in The Repeat Block Instruction example
Example 3-8	Modified text following Destination Register Conflict Resolved example
ADDF32 RaH, #16FHi, RbH ADDF32 RaH, RbH, #16FHi	Modified operand for instruction ADDF32 RaH, #16FHi, RbH. Updated the description.
ADDF32 RdH, ReH, RfH    MOV32 RaH, mem32	Modified the instruction ADDF32 RdH, ReH, RfH    MOV32 RaH, mem32
CMPF32 RaH, #16FHi	Modified the CMPF32 RaH, #16FHi instruction
CMPF32 RaH, #0.0	Modified the CMPF32 RaH, #0.0 instruction
F32TOI32 RaH, RbH	Modified the F32TO132 RaH, RbH instruction
F32TOUI32 RaH, RbH	Modified the F32TOUI32 RaH, RbH instruction
I16TOF32 RaH, RbH	Modified the I16TOF32 RaH, RbH instruction
MACF32 R3H, R2H, RdH, ReH	Modified the MACF32 R3H, R2H, RdH, ReH, RfH instruction
MAXF32 RaH, #16FHi	Modifed the syntax of the immediate operand. Modified the desciption.
MINF32 RaH, #16FHi	Modified the syntax of the immediate operand. Modified the description.
MINF32 RaH, RbH	Modified the MINF32 RaH, RbH instruction
MOV16 mem16, RaH	Modified the MOV16 mem16, RaH instruction
MOV32 loc32, *(0:16bitAddr)	Modified the MOV32 loc32, #(0:16bitAddr) instruction
MOV32 mem32, RaH	Modified the MOV32 mem32, RaH instruction
MOV32 mem32, STF	Modified the MOV32 mem32, STF instruction
MOV32 RaH, XT	Modified the MOV32 RaH, XT instruction
MOVF32 RaH, #32F	Added the MOV32 RaH, #32 instruction
MOVI32 RaH, #32FHex	Added the MOVI32 RaH, #32FHex instruction
MOVIZ RaH, #16FHiHex	Modified the syntax for the immediate operand. Modified the description. Modified the example.
MOVIZF32 RaH, #16FHi	Modified the syntax for the immediate operand. Modified the description.
MOVXI RaH, #16FLo	Modified the MOVXI RaH, #16FLo instruction. Modified the syntax of the immediate operand. Modified the description.
MPYF32 RaH, #16FHi, RbH MPYF32 RaH, RbH, #16FHi	Modified the syntax of the immediate operand. Modified the instruction description.
MPYF32 RdH, ReH, RfH	Modified the MPYF32 RdH, ReH, RfH   MOV32 RaH, mem32 instruction
SAVE FLAG, VALUE	Modified the SAVE FLAG, VALUE instruction
SUBF32 RaH, #16FHi, RbH	Modified the syntax for the immediate operand. Modified the description.

### Table A-1. Technical Changes Made in This Revision



Location	Additions, Deletions, Modifications
SUBF32 RdH, ReH, RfH    MOV32 mem32, RaH	Modified the SUBF32, RdH, ReH, RfH   MOV32 mem32, RaH instruction
UI16TOF32 RaH, RbH	Modified the UI16TOF32 RaH, RbH instruction
UI32TOF32 RaH, RbH	Modified the UI32TOF32 RaH, RbH instruction
Globally	The syntax sections of the #16F, #16I and #immF32 immediate addressing modes modes were changed to #16FHi, #16FHiHex, and #16FLoHex to be more descriptive and consistent. The descriptions for instructions using these modes were updated for clarity.
Example 2-2	Changed first instruction in example
Table 4-1	Updated the operand nomenclature table
Section 2.1.2	Modified the register figure introduction and register figure
EINVF32 RaH, RbH	Modified the example
EISQRTF32 RaH, RbH	Modified the example
Chapter 4	Added instructions to the See Also area of various instructions

### Table A-1. Technical Changes Made in This Revision (continued)

#### **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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2008, Texas Instruments Incorporated