TMS320x2803x Piccolo Control Law Accelerator (CLA)

Reference Guide



Literature Number: SPRUGE6B May 2009-Revised May 2010



Prefa	ace		6
1	Contro	bl Law Accelerator (CLA) Overview	8
2	CLA Ir	nterface	0
	2.1	CLA Memory 1	0
	2.2	CLA Memory Bus 1	0
	2.3	Shared Peripherals and EALLOW Protection 1	1
	2.4	CLA Tasks and Interrupt Vectors 1	2
3	CLA C	onfiguration and Debug	3
	3.1	Building a CLA Application	3
	3.2	Typical CLA Initialization Sequence 1	3
	3.3	Debugging CLA Code 1	5
	3.4	CLA Illegal Opcode Behavior 1	6
	3.5	Resetting the CLA 1	6
4	Regist	er Set 1	7
	4.1	Register Memory Mapping 1	7
	4.2	Task Interrupt Vector Registers 1	8
	4.3	Configuration Registers	8
	4.4	Execution Registers 2	9
5	Pipelir	ne 3	2
	5.1	Pipeline Overview	2
	5.2	CLA Pipeline Alignment 3	2
	5.3	Parallel Instructions 3	5
6	Instru	ction Set 3	7
	6.1	Instruction Descriptions 3	7
	6.2	Addressing Modes and Encoding 3	9
	6.3	Instructions 4	1
Арре	endix A	CLA and CPU Arbitration 15	1
	A.1	CLA and CPU Arbitration 15	1
Appe	endix B	Revision History 15	4



List of Figures

1	CLA Block Diagram	9
2	Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register	18
3	Control Register (MCTL)	18
4	Memory Configuration Register (MMEMCFG)	20
5	CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)	21
6	Interrupt Enable Register (MIER)	22
7	Interrupt Flag Register (MIFR)	23
8	Interrupt Overflow Flag Register (MIOVF)	24
9	Interrupt Run Status Register (MIRUN)	25
10	Interrupt Force Register (MIFRC)	26
11	Interrupt Flag Clear Register (MICLR)	27
12	Interrupt Overflow Flag Clear Register (MICLROVF)	28
13	Program Counter (MPC)	29
14	CLA Status Register (MSTF)	29

List of Tables

1	CLA Module Control and Status Register Set	17
2	Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Field Descriptions	18
3	Control Register (MCTL) Field Descriptions	19
4	Memory Configuration Register (MMEMCFG) Field Descriptions	20
5	Peripheral Interrupt Source Select 1 (MPISRCSEL1) Register Field Descriptions	21
6	Interrupt Enable Register (MIER) Field Descriptions	22
7	Interrupt Flag Register (MIFR) Field Descriptions	23
8	Interrupt Overflow Flag Register (MIOVF) Field Descriptions	24
9	Interrupt Run Status Register (MIRUN) Field Descriptions	25
10	Interrupt Force Register (MIFRC) Field Descriptions	26
11	Interrupt Flag Clear Register (MICLR) Field Descriptions	27
12	Interrupt Overflow Flag Clear Register (MICLROVF) Field Descriptions	28
13	Program Counter (MPC) Field Descriptions	29
14	CLA Status (MSTF) Register Field Descriptions	30
15	Write Followed by Read - Read Occurs First	33
16	Write Followed by Read - Write Occurs First	33
17	ADC to CLA Early Interrupt Response	35
18	Operand Nomenclature	37
19	INSTRUCTION dest, source1, source2 Short Description	38
20	Addressing Modes	39
21	Shift Field Encoding	39
22	Condition Field Encoding	40
23	Instructions	41
24	Pipeline Activity For MBCNDD, Branch Not Taken	56
25	Pipeline Activity For MBCNDD, Branch Taken	56
26	Pipeline Activity For MCCNDD, Call Not Taken	62
27	Pipeline Activity For MCCNDD, Call Taken	62
28	Pipeline Activity For MMOV16 MARx, MRa , #16I	94
29	Pipeline Activity For MMOV16 MAR0/MAR1, mem16	97
30	Pipeline Activity For MMOVI16 MAR0/MAR1, #16I	110
31	Pipeline Activity For MRCNDD, Return Not Taken	132



www.ti.com									
32	Pipeline Activity For MRCNDD, Return Taken	132							
33	Pipeline Activity For MSTOP	135							
34	Revisions to this Document	154							



The C28x Control Law Accelerator (CLA) is an independent, fully-programmable, 32-bit floating-point math processor that brings concurrent control-loop execution to the C28x family. The low interrupt-latency of the CLA allows it to read ADC samples "just-in-time." This significantly reduces the ADC sample to output delay to enable faster system response and higher MHz control loops. By using the CLA to service time-critical control loops, the main CPU is free to perform other system tasks such as communications and diagnostics. This document provides an overview of the architectural structure and instruction set of the C28x Control Law Accelerator.

The Control Law Accelerator module described in this reference guide is a Type 0 CLA. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (<u>SPRU566</u>) for a list of all devices with a CLA module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type. This document describes the architecture, pipeline, instruction set, and interrupts of the C28x Control Law Accelerator.

About This Manual

The TMS320C2000[™] is part of the TMS320[™] 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:

SPRS584 — TMS320F28032, TMS320F28033, TMS320F28034, TMS320F28035 Piccolo Microcontrollers Data Manual contains the pinout, signal descriptions, as well as electrical and timing specifications for the 2803x devices.

<u>SPRZ295</u> — TMS320F28032, TMS320F28033, TMS320F28034, TMS320F28035 Piccolo MCU Silicon Errata describes known advisories on silicon and provides workarounds.

CPU User's Guides—

SPRU430 — TMS320C28x CPU and Instruction Set Reference Guide 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.

Peripheral Guides—

<u>SPRUGL8</u> — TMS320x2803x Piccolo System Control and Interrupts Reference Guide describes the various interrupts and system control features of the 2803x microcontrollers (MCUs).

- <u>SPRU566</u> TMS320x28xx, 28xxx DSP Peripheral Reference Guide describes the peripheral reference guides of the 28x digital signal processors (DSPs).
- SPRUGO0 TMS320x2803x Piccolo Boot ROM Reference Guide describes the purpose and features of the boot loader (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.
- <u>SPRUGE6</u> TMS320x2803x Piccolo Control Law Accelerator (CLA) Reference Guide describes the operation of the Control Law Accelerator (CLA).
- <u>SPRUGE2</u> TMS320x2803x Piccolo Local Interconnect Network (LIN) Module Reference Guide describes the operation of the Local Interconnect Network (LIN) Module.
- <u>SPRUFK8</u> TMS320x2803x Piccolo Enhanced Quadrature Encoder Pulse (eQEP) Reference Guide describes the operation of the Enhanced Quadrature Encoder Pulse (eQEP).
- <u>SPRUGL7</u> TMS320x2803x Piccolo Enhanced Controller Area Network (eCAN) Reference Guide describes the operation of the Enhanced Controller Area Network (eCAN).
- SPRUGE5 TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide describes how to configure and use the on-chip ADC module, which is a 12-bit pipelined ADC.
- <u>SPRUGE9</u> TMS320x2802x, 2803x Piccolo 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.
- <u>SPRUGE8</u> TMS320x2802x, 2803x Piccolo High-Resolution Pulse Width Modulator (HRPWM) describes the operation of the high-resolution extension to the pulse width modulator (HRPWM).
- <u>SPRUGH1</u> TMS320x2802x, 2803x Piccolo Serial Communications Interface (SCI) Reference Guide describes how to use the SCI.
- <u>SPRUFZ8</u> TMS320x2802x, 2803x Piccolo Enhanced Capture (eCAP) Module Reference Guide describes the enhanced capture module. It includes the module description and registers.
- <u>SPRUG71</u> TMS320x2802x, 2803x Piccolo 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.
- SPRUFZ9 TMS320x2802x, 2803x Piccolo Inter-Integrated Circuit (I2C) Reference Guide describes the features and operation of the inter-integrated circuit (I2C) module.

Tools Guides—

- SPRU513 TMS320C28x Assembly Language Tools v5.0.0 User's Guide 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 TMS320C28x Optimizing C/C++ Compiler v5.0.0 User's Guide describes the TMS320C28x[™] C/C++ compiler. This compiler accepts ANSI standard C/C++ source code and produces TMS320 DSP assembly language source code for the TMS320C28x device.
- SPRU608 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[™] core.

TMS320C28x, C28x are trademarks of Texas Instruments.



TMS320x2803x Piccolo Control Law Accelerator (CLA)

The C28x Control Law Accelerator (CLA) is an independent, fully-programmable, 32-bit floating-point math processor that brings concurrent control-loop exceuction to the C28x family. The low interrupt-latency of the CLA allows it to read ADC samples "just-in-time". This significantly reduces the ADC sample to output delay to enable faster system response and higher MHz control loops. By using the CLA to service time-critical control loops, the main CPU is free to perform other system tasks such as communications and diagnostics. This chapter provides an overview of the arcitectural structure and components of the C28x Control Law Accelerator.

1 Control Law Accelerator (CLA) Overview

The control law accelerator extends the capabilities of the C28x CPU by adding parallel processing. Time-critical control loops serviced by the CLA can achieve low ADC sample to output delay. Thus, the CLA enables faster system response and higher frequency control loops. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently. The following is a list of major features of the CLA.

- Clocked at the same rate as the main CPU (SYSCLKOUT).
- An independent architecture allowing CLA algorithm execution independent of the main C28x CPU.
 - Complete bus architecture:
 - Program address bus and program data bus
 - Data address bus, data read bus and data write bus
 - Independent eight stage pipeline.
 - 12-bit program counter (MPC)
 - Four 32-bit result registers (MR0-MR3)
 - Two 16-bit auxiliary registers (MAR0, MAR1)
 - Status register (MSTF)
- Instruction set includes:
 - IEEE single-precision (32-bit) floating point math operations
 - Floating-point math with parallel load or store
 - Floating-point multiply with parallel add or subtract
 - 1/X and 1/sqrt(X) estimations
 - Data type conversions.
 - Conditional branch and call
 - Data load/store operations
- The CLA program code can consist of up to eight tasks or interrupt service routines.
 - The start address of each task is specified by the MVECT registers.
 - No limit on task size as long as the tasks fit within the CLA program memory space.
 - One task is serviced at a time through to completion. There is no nesting of tasks.
 - Upon task completion a task-specific interrupt is flagged within the PIE.
 - When a task finishes the next highest-priority pending task is automatically started.
- Task trigger mechanisms:
 - C28x CPU via the IACK instruction
 - Task1 to Task7: the corresponding ADC or ePWM module interrupt. For example:
 - Task1: ADCINT1 or EPWM1_INT



- Task2: ADCINT2 or EPWM2_INT
- Task7: ADCINT7 or EPWM7_INT
- Task8: ADCINT8 or by CPU Timer 0.
- Memory and Shared Peripherals:
 - Two dedicated message RAMs for communication between the CLA and the main CPU.
 - The C28x CPU can map CLA program and data memory to the main CPU space or CLA space.
 - The CLA has direct access to the ePWM+HRPWM, Comparator and ADC Result registers.





2 CLA Interface

This chapter describes how the C28x main CPU can interface to the CLA and vice versa.

2.1 CLA Memory

The CLA can access three types of memory: program, data and message RAMs. The behavior and arbitration for each type of memory is described in detail in Appendix A.

CLA Program Memory

At reset memory designated for CLA program is mapped to the main CPU memory and is treated like any other memory block. While mapped to CPU space, the main CPU can copy the CLA program code into the memory block. During debug the block can also be loaded directly by Code Composer Studio. Once the memory is initialized with CLA code, the main CPU maps it to the CLA program space by writing a 1 to the MMEMCFG[PROGE] bit. When mapped to the CLA program space, the block can only be accessed by the CLA for fetching opcodes. The main CPU can only perform debugger accesses when the CLA is either halted or idle. If the CLA is executing code, then all debugger accesses are blocked and the memory reads back all 0x0000.

CLA program memory is protected by the code security module. All CLA program fetches are performed as 32-bit read operations and all opcodes must be aligned to an even address. Since all CLA opcodes are 32-bits, this alignment naturally occurs.

CLA Data Memory

There are two CLA data memory blocks on the device. At reset, both blocks are mapped to the main CPU memory space and treated by the CPU like any other memory block. While mapped to CPU space, the main CPU can initialize the memory with data tables and coefficients for the CLA to use.

Once the memory is initialized with CLA data the main CPU maps it to the CLA space. Each block can be individually mapped via the MMEMCFG[RAM0E] and MMEMCFG[RAM1E] bits. When mapped to the CLA data space, the memory can be accessed only by the CLA for data operations. The main CPU can only perform debugger accesses in this mode.

Both CLA data RAMs are protected by the code security module and emulation code security logic.

• CLA Shared Message RAMs

There are two small memory blocks for data sharing and communication between the CLA and the main CPU. The message RAMs are always mapped to both CPU and CLA memory spaces and are protected by the code security module. The message RAMs allow data accesses only; no program fetches can be performed.

- CLA to CPU Message RAM

The CLA can use this block to pass data to the main CPU. This block is both readable and writable by the CLA. This block is also readable by the main CPU but writes by the main CPU are ignored.

CPU to CLA Message RAM

The main CPU can use this block to pass data and messages to the CLA. This message RAM is both readable and writable by the main CPU. The CLA can perform reads but writes by the CLA are ignored.

2.2 CLA Memory Bus

The CLA has dedicated bus architecture similar to that of the C28x CPU where there is a program read, data read and data write bus. Thus there can be simultaneous instruction fetch, data read and data write in a single cycle. Like the C28x CPU, the CLA expects memory logic to align any 32-bit read or write to an even address. If the address-generation logic generates an odd address, the CLA will begin reading or writing at the previous even address. This alignment does not affect the address values generated by the address-generation logic.

CLA Program Bus

The CLA program bus has a access range of 2048 32-bit instructions. Since all CLA instructions are 32-bits, this bus always fetches 32-bits at a time and the opcodes must be even word aligned. The amount of program space available for the CLA is device dependent as described in the device-specific data manual.

CLA Data Read Bus

The CLA data read bus has a 64K x 16 address range. The bus can perform 16 or 32-bit reads and



will automatically stall if there are memory access conflicts. The data read bus has access to both the message RAMs, CLA data memory and the ePWM, HRPWM, Comparator and ADC result registers.

CLA Data Write Bus

The CLA data write bus has a 64K x 16 address range. This bus can perform 16 or 32-bit writes. The bus will automatically stall if there are memory access conflicts. The data write bus has access to the CLA to CPU message RAM, CLA data memory and the ePWM, HRPWM, and Comparator registers.

2.3 Shared Peripherals and EALLOW Protection

The ePWM, HRPWM, Comparator, and ADC result registers can be accessed by both the CLA and the main CPU. Appendix A describes in detail the CLA and CPU arbitration when both access these registers.

Several peripheral control registers are protected from spurious 28x CPU writes by the EALLOW protection mechanism. These same registers are also protected from spurious CLA writes. The EALLOW bit in the main CPU status register 1 (ST1) indicates the state of protection for the main CPU. Likewise the MEALLOW bit in the CLA status register (MSTF) indicates the state of write protection for the CLA. The MEALLOW CLA instruction enables write access by the CLA to EALLOW protected registers. Likewise the MEDIS CLA instruction will disable write access. This way the CLA can enable/disable write access independent of the main CPU.

The 2803x ADC offers the option to generate an early interrupt pulse when the ADC begins conversion. If this option is used to start a ADC triggered CLA task then the 8th instruction can read the result as soon as the conversion completes. The CLA pipeline activity for this scenario is shown in Section 5.



2.4 CLA Tasks and Interrupt Vectors

The CLA program code is divided up into tasks or interrupt service routines. Tasks do not have a fixed starting location or length. The CLA program memory can be divided up as desired. The CLA knows where a task begins by the content of the associated interrupt vector (MVECT1 to MVECT8) and the end is indicated by the MSTOP instruction.

The CLA supports 8 tasks. Task 1 has the highest priority and task 8 has the lowest priority. A task can be requested by a peripheral interrupt or by software:

Peripheral interrupt trigger

Each task has specific interrupt sources that can trigger it. Configure the MPISRCSEL1 register to select from the potential sources. For example, task 1 (MVECT1) can be triggered by ADCINT1 or EPWM1_INT as specified in MPISRCSEL1[PERINT1SEL]. You can not, however, trigger task 1 directly using EPWM2_INT. If you need to trigger a task using EPWM2_INT then the best solution is to use task 2 (MVECT2). Another possible solution is to take EPWM2_INT with the main CPU and trigger a task with software.

To disable the peripheral from sending an interrupt request to the CLA set the PERINT1SEL option to no interrupt.

• Software trigger

Tasks can also be started by the main CPU software writing to the MIFRC register or by the IACK instruction. Using the IACK instruction is more efficient because it does not require you to issue an EALLOW to set MIFR bits. Set the MCTL[IACKE] bit to enable the IACK feature. Each bit in the operand of the IACK instruction corresponds to a task. For example IACK #0x0001 will set bit 0 in the MIFR register to start task 1. Likewise IACK #0x0003 will set bits 0 and 1 in the MIFR register to start task 1 and task 2.

The CLA has its own fetch mechanism and can run and execute a task independent of the main CPU. Only one task is serviced at a time; there is no nesting of tasks. The task currently running is indicated in the MIRUN register. Interrupts that have been received but not yet serviced are indicated in the flag register (MIFR). If an interrupt request from a peripheral is received and that same task is already flagged, then the overflow flag bit is set. Overflow flags will remain set until they are cleared by the main CPU.

If the CLA is idle (no task is currently running) then the highest priority interrupt request that is both flagged (MIFR) and enabled (MIER) will start. The flow is as follows

- 1. The associated RUN register bit is set (MIRUN) and the flag bit (MIFR) is cleared.
- 2. The CLA begins execution at the location indicated by the associated interrupt vector (MVECTx). MVECT is an offset from the first program memory location.
- 3. The CLA executes instructions until the MSTOP instruction is found. This indicates the end of the task.
- 4. The MIRUN bit is cleared.
- 5. The task-specific interrupt to the PIE is issued. This informs the main CPU that the task has completed.
- 6. The CLA returns to idle.

Once a task completes the next highest-priority pending task is automatically serviced and this sequence repeats.



3 CLA Configuration and Debug

This section discusses the steps necessary to configure and debug the CLA.

3.1 Building a CLA Application

The Control Law Accelerator is programmed in CLA assembly code using the instructions described in Section 6. CLA assembly code can, and should, reside in the same project with C28x code. The only restriction is the CLA code must be in its own assembly section. This can be easily done using the .sect assembly directive. This does not prevent CLA and C28x code from being linked into the same memory region in the linker command file.

System and CLA initialization are performed by the main CPU. This would typically be done in C or C++ but can also include C28x assembly code. The main CPU will also copy the CLA code to the program memory and, if needed, initialize the CLA data RAM(s). Once system initialization is complete and the application begins, the CLA will service its interrupts using the CLA assembly code (or tasks). Concurrently the main CPU can perform other tasks.

The C2000 codegen tools V5.2.x and higher support CLA instructions when the following switch is set: -- $cla_support = cla0$.

3.2 Typical CLA Initialization Sequence

A typical CLA initialization sequence is performed by the main CPU as described in this section.

1. Copy CLA code into the CLA program RAM

The source for the CLA code can initially reside in the flash or a data stream from a communications peripheral or anywhere the main CPU can access it. The debugger can also be used to load code directly to the CLA program RAM during development.

2. Initialize CLA data RAM if necessary

Populate the CLA data RAM with any required data coefficients or constants.

3. Configure the CLA registers

Configure the CLA registers, but keep interrupts disabled until later (leave MIER == 0):

• Enable the CLA clock in the PCLKCR3 register.

PCLKCR3 register is defined in the device-specific system control and interrupts reference guide.

• Populate the CLA task interrupt vectors: MVECT1 to MVECT8.

Each vector needs to be initialized with the start address of the task to be executed when the CLA receives the associated interrupt. This address is an offset from the first address in CLA program memory. For example, 0x0000 corresponds to the first CLA program memory address.

Select the task interrupt sources

For each task select the interrupt source in the PERINT1SEL register. If a task is going to be generated by software, select no interrupt.

• Enable IACK to start a task from software if desired

To enable the IACK instruction to start a task set the MCTL[IACKE] bit. Using the IACK instruction avoids having to set and clear the EALLOW bit.

Map CLA data RAM(s) to CLA space if necessary

Map either or both of the data RAMs to the CLA space by writing a 1 to the MMEMCFG[RAM0E] and MMEMCFG[RAM1E] bits. After the memory is mapped to CLA space the main CPU cannot access it. Allow two SYSCLKOUT cycles between changing the map configuration of this memory and accessing it.

• Map CLA program RAM to CLA space

Map the CLA program RAM to CLA space by setting the MMEMCFG[PROGE] bit. After the memory is remapped to CLA space the main CPU will only be able to make debug accesses to the memory block. Allow two SYSCLKOUT cycles between changing the map configuration of these memories and accessing them.

4. Initialize the PIE vector table and registers

When a CLA task completes the associated interrupt in the PIE will be flagged. The CLA overflow and underflow flags also have associated interrupts within the PIE.

5. Enable CLA tasks/interrupts

Set appropriate bits in the interrupt enable register (MIER) to allow the CLA to service interrupts.

6. Initialize other peripherals

Initialize any peripherals (ePWM, ADC etc.) that will generate an interrupt to the CLA and be serviced by a CLA task.

The CLA is now ready to service interrupts and the message RAMs can be used to pass data between the CPU and the CLA. Typically mapping of the CLA program and data RAMs occurs only during the initialization process. If after some time the you want to re-map these memories back to CPU space then disable interrupts and make sure all tasks have completed by checking the MIRUN register. Always allow two SYSCLKOUT cycles when changing the map configuration of these memories and accessing them.





3.3 Debugging CLA Code

Debugging the CLA code is a simple process that occurs independently of the main CPU.

1. Insert a breakpoint in CLA code

Insert a CLA breakpoint (MDEBUGSTOP instruction) into the code where you want the CLA to halt, then rebuild and reload the code. Because the CLA does not flush its pipeline when you single-step, the MDEBUGSTOP instruction must be inserted as part of the code. The debugger cannot insert it as needed.

If CLA breakpoints are not enabled, then the MDEBUGSTOP will be ignored and is treated as a MNOP. The MDEBUGSTOP instruction can be placed anywhere in the CLA code as long as it is not within three instructions of a MBCNDD, MCCNDD, or MRCNDD instruction.

2. Enable CLA breakpoints

First, enable the CLA breakpoints in the debugger. In Code Composer Studio V3.3, this is done by connecting the CLA debug window (debug->connect). Breakpoints are disabled when this window is disconnected.

3. Start the task

There are three ways to start the task:

- The peripheral can assert an interrupt
- The main CPU can execute an IACK instruction, or
- You can manually write to the MIFRC register in the debugger window

When the task starts, the CLA will execute instructions until the MDEBUGSTOP is in the D2 phase of the pipeline. At this point, the CLA will halt and the pipeline will be frozen. The MPC register will reflect the address of the MDEBUGSTOP instruction.

4. Single-step the CLA code

Once halted, you can single-step the CLA code one cycle at a time. The behavior of a CLA single-step is different than the main C28x. When issuing a CLA single-step, the pipeline is clocked only one cycle and then again frozen. On the 28x CPU, the pipeline is flushed for each single-step.

You can also run to the next MDEBUGSTOP or to the end of the task. If another task is pending, it will automatically start when you run to the end of the task.

NOTE: When CLA program memory is mapped to the CLA memory space, a CLA fetch has higher priority than CPU debug reads. For this reason, it is possible for the CLA to permanently block CPU debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug that causes an infinite loop. To avoid locking up the main CPU, the program memory will return all 0x0000 for CPU debug reads when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access to CLA program memory can be performed.

If the CLA gets caught in a infinite loop, you can use a soft or hard reset to exit the condition. A debugger reset will also exit the condition.

There are special cases that can occur when single-stepping a task such that the program counter, MPC, reaches the MSTOP instruction at the end of the task.

MPC halts at or after the MSTOP with a task already pending

If you are single-stepping or halted in "task A" and "task B" comes in before the MPC reaches the MSTOP, then "task B" will start if you continue to step through the MSTOP instruction. Basically if "task B" is pending before the MPC reaches MSTOP in "task A" then there is no issue in "task B" starting and no special action is required.

• MPC halts at or after the MSTOP with no task pending

In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. If "task B" comes in at this point, it will be flagged in the MIFR register but it may or may not start if you continue to single-step through the MSTOP instruction of "task A." It depends on exactly when the new task comes in. To reliably start "task B" perform a soft reset and reconfigure the MIER bits. Once this is done, you can start single-stepping "task B."

This case can be handled slightly differently if there is control over when "task B" comes in (for example using the IACK instruction to start the task). In this case you have single-stepped or halted



CLA Configuration and Debug

in "task A" and the MPC has reached the MSTOP with no tasks pending. Before forcing "task B," run free to force the CLA out of the debug state. Once this is done you can force "task B" and continue debugging.

5. If desired, disable CLA breakpoints

In CCS V3.3 you can disable the CLA breakpoints by disconnecting the CLA debug window. Make sure to first issue a run or reset; otherwise, the CLA will be halted and no other tasks will start.

3.4 CLA Illegal Opcode Behavior

If the CLA fetches an opcode that does not correspond to a legal instruction, it will behave as follows:

- The CLA will halt with the illegal opcode in the D2 phase of the pipeline as if it were a breakpoint. This will occur whether CLA breakpoints are enabled or not.
- The CLA will issue the task-specific interrupt to the PIE.
- The MIRUN bit for the task will remain set.

Further single-stepping ignored once execution halts due to an illegal op-code. To exit this situation, issue either a soft or hard reset of the CLA as described in Section 3.5.

3.5 Resetting the CLA

There may be times when you need to reset the CLA. For example, during code debug the CLA may enter an infinite loop due to a code bug. The CLA has two types of resets: hard and soft. Both of these resets can be performed by the debugger or by the main CPU.

Hard Reset

Writing a 1 to the MCTL[HARDRESET] bit will perform a hard reset of the CLA. The behavior of a hard reset is the same as a system reset (via \overline{XRS} or the debugger). In this case all CLA configuration and execution registers will be set to their default state and CLA execution will halt.

Soft Reset

Writing a 1 to the MCTL[SOFTRESET] bit performs a soft reset of the CLA. If a task is executing it will halt and the associated MIRUN bit will be cleared. All bits within the interrupt enable (MIER) register will also be cleared so that no new tasks start.

4 Register Set

The CLA register set is independant from that of the main CPU. This chapter describes the CLA register set.

4.1 Register Memory Mapping

The table below describes the CLA module control and status register set.

		Size		CSM	
Name	Offset	(x16)	EALLOW	Protected	Description
					Task Interrupt Vectors
MVECT1	0x0000	1	Yes	Yes	Task 1 Interrupt Vector
MVECT2	0x0001	1	Yes	Yes	Task 2 Interrupt Vector
MVECT3	0x0002	1	Yes	Yes	Task 3 Interrupt Vector
MVECT4	0x0003	1	Yes	Yes	Task 4 Interrupt Vector
MVECT5	0x0004	1	Yes	Yes	Task 5 Interrupt Vector
MVECT6	0x0005	1	Yes	Yes	Task 6 Interrupt Vector
MVECT7	0x0006	1	Yes	Yes	Task 7 Interrupt Vector
MVECT8	0x0007	1	Yes	Yes	Task 8 Interrupt Vector
					Configuration Registers
MCTL	0x0010	1	Yes	Yes	Control Register
MMEMCFG	0x0011	1	Yes	Yes	Memory Configuration Register
MPISRCSEL1	0x0014	2	Yes	Yes	Peripheral Interrupt Source Select 1 Register
MIFR	0x0020	1	Yes	Yes	Interrupt Flag Register
MIOVF	0x0021	1	Yes	Yes	Interrupt Overflow Flag Register
MIFRC	0x0022	1	Yes	Yes	Interrupt Force Register
MICLR	0x0023	1	Yes	Yes	Interrupt Flag Clear Register
MICLROVF	0x0024	1	Yes	Yes	Interrupt Overflow Flag Clear Register
MIER	0x0025	1	Yes	Yes	Interrupt Enable Register
MIRUN	0x0026	1	Yes	Yes	Interrupt Run Status Register
					Execution Registers ⁽¹⁾
MPC	0x0028	1	-	Yes	CLA Program Counter
MAR0	0x0029	1	-	Yes	CLA Auxiliary Register 0
MAR1	0x002A	1	-	Yes	CLA Auxiliary Register 1
MSTF	0x002E	2	-	Yes	CLA Floating-Point Status Register
MR0	0x0030	2	-	Yes	CLA Floating-Point Result Register 0
MR1	0x0034	2	-	Yes	CLA Floating-Point Result Register 1
MR2	0x0038	2	-	Yes	CLA Floating-Point Result Register 2
MR3	0x003C	2	-	Yes	CLA Floating-Point Result Register 3

 Table 1. CLA Module Control and Status Register Set

⁽¹⁾ The main C28x CPU only has read access to the CLA execution registers for debug purposes. The main CPU cannot perform CPU or debugger writes to these registers.

Register Set



4.2 Task Interrupt Vector Registers

Each CLA interrupt has its own interrupt vector (MVECT1 to MVECT8). This interrupt vector points to the first instruction of the associated task. When a task begins, the CLA will start fetching instructions at the location indicated by the appropriate MVECT register .

4.2.1 Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register

The task interrupt vector registers (MVECT1/2/3/4/5/6/7/8) are is shown in Section 4.2.1 and described in Figure 2.

Figure 2. Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register

15	12	11	0
Res	erved	MVECT	
F	۲-0	R-0	

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

Table 2. Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾				
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.				
11-0	MVECT	0000 - 0FFF	Offset of the first instruction in the associated task from the start of CLA program space. The CLA will begin instruction fetches from this location when the specific task begins.				
			For example: If CLA program memory begins at CPU address 0x009000 and the code for task 5 begins at CPU address 0x009120, then MVECT5 should be initialized with 0x0120.				
			There is one MVECT register per task. Interrupt 1 uses MVECT1, interrupt 2 uses MVECT2 and so forth.				

⁽¹⁾ These registers are protected by EALLOW and the code security module.

4.3 Configuration Registers

The configuration registers are described here.

4.3.1 Control Register (MCTL)

The configuration control register (MCTL) is shown in Figure 3 and described in Table 3.

Figure 3. Control Register (MCTL)

15						8
		Reserved				
		R -0				
7			3	2	1	0
	Reserved			IACKE	SOFTRESET	HARDRESET
	R/W-0			R/W-0	R/W-0	R/W-0

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



Bits	Name	Value	Description ⁽¹⁾					
15-3	Reserved		Any writes to these bit(s) must always have a value of 0.					
2	IACKE		IACK enable					
		0	The CLA ignores the IACK instruction. (default)					
		1	Enable the main CPU to use the IACK #16bit instruction to set MIFR bits in the same manner as writing to the MIFRC register. Each bit in the operand, #16bit, corresponds to a bit in the MIFRC register. Using IACK has the advantage of not having to first set the EALLOW bit. This allows the main CPU to efficiently trigger a CLA task through software.					
			Examples IACK #0x0001 Write a 1 to MIFRC bit 0 to force task 1					
			IACK #0x0003 Write a 1 to MIFRC bit 0 and 1 to force task 1 and task 2					
1	SOFTRESET		Soft Reset					
		0	This bit always reads back 0 and writes of 0 are ignored.					
		1	Writing a 1 will cause a soft reset of the CLA. This will stop the current task, clear the MIRUN flag and clear all bits in the MIER register. After a soft reset you must wait at least 1 SYSCLKOUT cycle before reconfiguring the MIER bits. If these two operations are done back-to-back then the MIER bits will not get set.					
0	HARDRESET		Hard Reset					
		0	This bit always reads back 0 and writes of 0 are ignored.					
		1	Writing a 1 will cause a hard reset of the CLA. This will set all CLA registers to their default state.					

Table 3. Control Register (MCTL) Field Descriptions

⁽¹⁾ This register is protected by EALLOW and the code security module.



...

4.3.2 Memory Configuration Register (MMEMCFG)

The MMEMCFG register is used to map the CLA program and data RAMs to either the CPU or the CLA memory space. Typically mapping of the CLA program and data RAMs occurs only during the initialization process. If after some time the you want to re-map these memories back to CPU space then disable interrupts (MIER) and make sure all tasks have completed by checking the MIRUN register. Allow two SYSCLKOUT cycles between changing the map configuration of these memories and accessing them. Refer to Section A.1.3 for CLA and CPU access arbitration details.

Figure 4. Memory Configuration Register (MMEMCFG)

15							8
			Rese	rved			
	R -0						
7	6	5	4	3		1	0
Reserved		RAM1E	RAM0E		Reserved		PROGE
R	·0	R/W-0	R/W-0		R-0		R/W-0
LECEND: DAM Bood AMinter D. Dood only n. volus ofter reset							

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

Table 4. Memory Configuration Register (MMEMCFG) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾		
15-6	Reserved		Any writes to these bit(s) must always have a value of 0.		
5	RAM1E		CLA Data RAM 1 Enable		
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory.		
		0	The CLA data SARAM block 1 is mapped to the main CPU program and data space. CLA reads will return zero. (default)		
		1	The CLA data SARAM block 1 is mapped to CLA data space. The main CPU can only make debug accesses to this block.		
4	RAM0E		CLA Data RAM 0 Enable		
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory.		
		0	The CLA data SARAM block 0 is mapped to the main CPU program and data space. CLA reads will return zero. (default)		
		1	The CLA data SARAM block 0 is mapped to CLA data space. The main CPU can only make debug accesses to this block.		
3 - 1	Reserved		Any writes to these bit(s) must always have a value of 0.		
0	PROGE		CLA Program Space Enable		
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory.		
		0	CLA program SARAM is mapped to the main CPU program and data space. If the CLA attempts a program fetch the result will be the same as an illegal opcode fetch as described in Section 3.4. (default)		
		1	CLA program SARAM is mapped to the CLA program space. The main CPU can only make debug accesses to this block.		
			In this state a CLA fetch has higher priority than CPU debug reads. It is, therefore, possible for the CLA to permanently block debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug. To avoid this issue, the program memory will return all 0x0000 for CPU debug reads (ignore writes) when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access can be performed.		

⁽¹⁾ This register is protected by EALLOW and the code security module.

4.3.3 CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)

Each task has specific peripherals that can start it. For example, Task2 can be started by ADCINT2 or EPWM2_INT. To configure which of the possible peripherals will start a task configure the MPISRCSEL1 register shown in Figure 5. Choosing the option "no interrupt source" means that only the main CPU software will be able to start the given task.

Figure 5. CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)										
31	28	27	24	23	20	19	16			
PE	RINT8SEL	PERINT7SEL		PERINT6SEL		PERINT5SEL				
	R/W-0	R/W-0		R/W-0		R/W-0				
15	12	11	8	7	4	3	0			
PE	RINT4SEL	PER	PERINT3SEL		PERINT2SEL		RINT1SEL			
	R/W-0	F	R/W-0	R/W-0		F	R/W-0			

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

Table 5. Peripheral Interrupt Source Select 1 (MPISRCSEL1) Register Field Descriptions

Bits	Field	Value (1)	Description (2)
31 - 28	PERINT8SEL		Task 8 Peripheral Interrupt Input Select
		0000	ADCINT8 is the input for interrupt task 8. (default)
		0010	CPU Timer 0 is the input for interrupt task 8.
		xxx1	No interrupt source for task 8.
27 - 24	PERINT7SEL		Task 7 Peripheral Interrupt Input Select
		0000	ADCINT7 is the input for interrupt task 7. (default)
		0010	ePWM7 is the input for interrupt task 7. (EPWM7_INT)
		xxx1	No interrupt source for task 7.
23 - 20	PERINT6SEL		Task 6 Peripheral Interrupt Input Select
		0000	ADCINT6 is the input for interrupt task 6. (default)
		0010	ePWM6 is the input for interrupt task 6. (EPWM6_INT)
		xxx1	No interrupt source for task 6.
19 - 16	PERINT5SEL		Task 5 Peripheral Interrupt Input Select
		0000	ADCINT5 is the input for interrupt task 5. (default)
		0010	ePWM5 is the input for interrupt task 5. (EPWM5_INT)
		xxx1	No interrupt source for task 5.
15 - 12	PERINT4SEL		Task 4 Peripheral Interrupt Input Select
		0000	ADCINT4 is the input for interrupt task 4. (default)
		0010	ePWM4 is the input for interrupt task 4. (EPWM4_INT)
		xxx1	No interrupt source for task 4.
11 - 8	PERINT3SEL		Task 3 Peripheral Interrupt Input Select
		0000	ADCINT3 is the input for interrupt task 3. (default)
		0010	ePWM3 is the input for interrupt task 3. (EPWM3_INT)
		xxx1	No interrupt source for task 3.
7 - 4	PERINT2SEL		Task 2 Peripheral Interrupt Input Select
		0000	ADCINT2 is the input for interrupt task 2. (default)
		0010	ePWM2 is the input for interrupt task 2. (EPWM2_INT)
		xxx1	No interrupt source for task 2.
3 - 0	PERINT1SEL		Task 1Peripheral Interrupt Input Select
		0000	ADCINT1 is the input for interrupt task 1. (default)
		0010	ePWM1 is the input for interrupt task 1. (EPWM1_INT)
		xxx1	No interrupt source

⁽¹⁾ All values not shown are reserved.

⁽²⁾ This register is protected by EALLOW and the code security module.

. -

4.3.4 Interrupt Enable Register (MIER)

Setting the bits in the interrupt enable register (MIER) allow an incoming interrupt or main CPU software to start the corresponding CLA task. Writing a 0 will block the task, but the interrupt request will still be latched in the flag register (MIFLG). Setting the MIER register bit to 0 while the corresponding task is executing will have no effect on the task. The task will continue to run until it hits the MSTOP instruction.

When a soft reset is issued, the MIER bits are cleared. There should always be at least a 1 SYSCLKOUT delay between issuing the soft reset and reconfiguring the MIER bits.

Figure 6. Interrupt Enable Register (MIER)

15							8				
Reserved											
R -0											
7	6	5	4	3	2	1	0				
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1				
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0				

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

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Enable
		0	Task 8 interrupt is disabled. (default)
		1	Task 8 interrupt is enabled.
6	INT7		Task 7 Interrupt Enable
		0	Task 7 interrupt is disabled. (default)
		1	Task 7 interrupt is enabled.
5	INT6		Task 6 Interrupt Enable
		0	Task 6 interrupt is disabled. (default)
		1	Task 6 interrupt is enabled.
4	INT5		Task 5 Interrupt Enable
		0	Task 5 interrupt is disabled. (default)
		1	Task 5 interrupt is enabled.
3	INT4		Task 4 Interrupt Enable
		0	Task 4 interrupt is disabled. (default)
		1	Task 4 interrupt is enabled.
2	INT3		Task 3 Interrupt Enable
		0	Task 3 interrupt is disabled. (default)
		1	Task 3 interrupt is enabled.
1	INT2		Task 2 Interrupt Enable
		0	Task 2 interrupt is disabled. (default)
		1	Task 2 interrupt is enabled.
0	INT1		Task 1 Interrupt Enable
		0	Task 1 interrupt is disabled. (default)
		1	Task 1 interrupt is enabled.

Table 6. Interrupt Enable Register (MIER) Field Descriptions

⁽¹⁾ This register is protected by EALLOW and the code security module.

4.3.5 Interrupt Flag Register (MIFR)

Each bit in the interrupt flag register corresponds to a CLA task. The corresponding bit is automatically set when the task request is received from the peripheral interrupt. The bit can also be set by the main CPU writing to the MIFRC register or using the IACK instruction to start the task. To use the IACK instruction to begin a task first enable this feature in the MCTL register. If the bit is already set when a new peripheral interrupt is received, then the corresponding overflow bit will be set in the MIOVF register.

The corresponding MIFR bit is automatically cleared when the task begins execution. This will occur if the interrupt is enabled in the MIER register and no other higher priority task is pending. The bits can also be cleared manually by writing to the MICLR register. Writes to the MIFR register are ignored.

15							8					
	Reserved											
R -0												
7	6	5	4	3	2	1	0					
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1					
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0					

Figure 7. Interrupt Flag Register (MIFR)

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

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Flag
		0	A task 8 interrupt is currently not flagged. (default)
		1	A task 8 interrupt has been received and is pending execution.
6	INT7		Task 7 Interrupt Flag
		0	A task 7 interrupt is currently not flagged. (default)
		1	A task 7 interrupt has been received and is pending execution.
5	INT6		Task 6 Interrupt Flag
		0	A task 6 interrupt is currently not flagged. (default)
		1	A task 6 interrupt has been received and is pending execution.
4	INT5		Task 5 Interrupt Flag
		0	A task 5 interrupt is currently not flagged. (default)
		1	A task 5 interrupt has been received and is pending execution.
3	INT4		Task 4 Interrupt Flag
		0	A task 4 interrupt is currently not flagged. (default)
		1	A task 4 interrupt has been received and is pending execution.
2	INT3		Task 3 Interrupt Flag
		0	A task 3 interrupt is currently not flagged. (default)
		1	A task 3 interrupt has been received and is pending execution.
1	INT2		Task 2 Interrupt Flag
		0	A task 2 interrupt is currently not flagged. (default)
		1	A task 2 interrupt has been received and is pending execution.
0	INT1		Task 1 Interrupt Flag
		0	A task 1 interrupt is currently not flagged. (default)
		1	A task 1 interrupt has been received and is pending execution.

Table 7. Interrupt Flag Register (MIFR) Field Descriptions

⁽¹⁾ This register is protected by the code security module.

4.3.6 Interrupt Overflow Flag Register (MIOVF)

Each bit in the overflow flag register corresponds to a CLA task. The bit is set when an interrupt overflow event has occurred for the specific task. An overflow event occurs when the MIFR register bit is already set when a new interrupt is received from a peripheral source. The MIOVF bits are only affected by peripheral interrupt events. They do not respond to a task request by the main CPU IACK instruction or by directly setting MIFR bits. The overflow flag will remain latched and can only be cleared by writing to the overflow flag clear (MICLROVF) register. Writes to the MIOVF register are ignored.

15							8			
Reserved										
R -0										
7	6	5	4	3	2	1	0			
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1			
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0			

Figure 8. Interrupt Overflow Flag Register (MIOVF)

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

Table 8. Interrupt Overflow Flag Register (MIOVF) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Overflow Flag
		0	A task 8 interrupt overflow has not occurred. (default)
		1	A task 8 interrupt overflow has occurred.
6	INT7		Task 7 Interrupt Overflow Flag
		0	A task 7 interrupt overflow has not occurred. (default)
		1	A task 7 interrupt overflow has occurred.
5	INT6		Task 6 Interrupt Overflow Flag
		0	A task 6 interrupt overflow has not occurred. (default)
		1	A task 6 interrupt overflow has occurred.
4	INT5		Task 5 Interrupt Overflow Flag
		0	A task 5 interrupt overflow has not occurred. (default)
		1	A task 5 interrupt overflow has occurred.
3	INT4		Task 4 Interrupt Overflow Flag
		0	A task 4 interrupt overflow has not occurred. (default)
		1	A task 4 interrupt overflow has occurred.
2	INT3		Task 3 Interrupt Overflow Flag
		0	A task 3 interrupt overflow has not occurred. (default)
		1	A task 3 interrupt overflow has occurred.
1	INT2		Task 2 Interrupt Overflow Flag
		0	A task 2 interrupt overflow has not occurred. (default)
		1	A task 2 interrupt overflow has occurred.
0	INT1		Task 1 Interrupt Overflow Flag
		0	A task 1 interrupt overflow has not occurred. (default)
		1	A task 1 interrupt overflow has occurred.

⁽¹⁾ This register is protected by the code security module.

4.3.7 Interrupt Run Status Register (MIRUN)

The interrupt run status register (MIRUN) indicates which task is currently executing. Only one MIRUN bit will ever be set to a 1 at any given time. The bit is automatically cleared when the task competes and the respective interrupt is fed to the peripheral interrupt expansion (PIE) block of the device. This lets the main CPU know when a task has completed. The main CPU can stop a currently running task by writing to the MCTL[SOFTRESET] bit. This will clear the MIRUN flag and stop the task. In this case no interrupt will be generated to the PIE.

Figure 9. Interrupt Run Status Register (MIRUN)

15							8				
Reserved											
R -0											
7	6	5	4	3	2	1	0				
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1				
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0				
	-										

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

Table 9. Interrupt Run Status Register (MIRUN) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Run Status
		0	Task 8 is not executing. (default)
		1	Task 8 is executing.
6	INT7		Task 7 Run Status
		0	Task 7 is not executing. (default)
		1	Task 7 is executing.
5	INT6		Task 6 Run Status
		0	Task 6 is not executing. (default)
		1	Task 6 is executing.
4	INT5		Task 5 Run Status
		0	Task 5 is not executing. (default)
		1	Task 5 is executing.
3	INT4		Task 4 Run Status
		0	Task 4 is not executing. (default)
		1	Task 4 is executing.
2	INT3		Task 3 Run Status
		0	Task 3 is not executing. (default)
		1	Task 3 is executing.
1	INT2		Task 2 Run Status
		0	Task 2 is not executing. (default)
		1	Task 2 is executing.
0	INT1		Task 1 Run Status
		0	Task 1 is not executing. (default)
		1	Task 1 is executing.

This register is protected by the code security module. (1)

4.3.8 Interrupt Force Register (MIFRC)

The interrupt force register can be used by the main CPU to start tasks through software. Writing a 1 to a MIFRC bit will set the corresponding bit in the MIFR register. Writes of 0 are ignored and reads always return 0. The IACK #16bit operation can also be used to start tasks and has the same effect as the MIFRC register. To enable IACK to set MIFR bits you must first set the MCTL[IACKE] bit. Using IACK has the advantage of not having to first set the EALLOW bit. This allows the main CPU to efficiently trigger CLA tasks through software.

Figure 10. Interrupt Force Register (MIFRC) 15 8 Reserved R -0 7 6 5 4 3 2 1 0 INT8 INT7 INT6 INT5 INT4 INT2 INT3 INT1 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0

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

Table 10. Interrupt Force Register (MIFRC) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 8 interrupt.
6	INT7		Task 7 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 7 interrupt.
5	INT6		Task 6 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 6 interrupt.
4	INT5		Task 5 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 5 interrupt.
3	INT4		Task 4 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 4 interrupt.
2	INT3		Task 3 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 3 interrupt.
1	INT2		Task 2 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 2 interrupt.
0	INT1		Task 1 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 1 interrupt.
(1) TI	and states after some		

¹⁾ This register is protected by EALLOW and the code security module.



4.3.9 Interrupt Flag Clear Register (MICLR)

Normally bits in the MIFR register are automatically cleared when a task begins. The interrupt flag clear register can be used to instead manually clear bits in the interrupt flag (MIFR) register. Writing a 1 to a MICLR bit will clear the corresponding bit in the MIFR register. Writes of 0 are ignored and reads always return 0.

15		-	-	-		-	8				
Reserved											
R -0											
7	6	5	4	3	2	1	0				
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1				
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0				

Figure 11. Interrupt Flag Clear Register (MICLR)

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

Table 11. Interrupt Flag Clear Register (MICLR) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 8 interrupt flag.
6	INT7		Task 7 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 7 interrupt flag.
5	INT6		Task 6 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 6 interrupt flag.
4	INT5		Task 5 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 5 interrupt flag.
3	INT4		Task 4 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 4 interrupt flag.
2	INT3		Task 3 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 3 interrupt flag.
1	INT2		Task 2 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 2 interrupt flag.
0	INT1		Task 1 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 1 interrupt flag.

⁽¹⁾ This register is protected by EALLOW and the code security module.



4.3.10 Interrupt Overflow Flag Clear Register (MICLROVF)

Overflow flag bits in the MIOVF register are latched until manually cleared using the MICLROVF register. Writing a 1 to a MICLROVF bit will clear the corresponding bit in the MIOVF register. Writes of 0 are ignored and reads always return 0.

Figure 12. Interrupt Overflow Flag Clear Register (MICLROVF)

15							8
			Rese	erved			
			R	-0			
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
		- · ·					

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

Table 12. Interrupt Overflow Flag Clear Register (MICLROVF) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 8 interrupt overflow flag.
6	INT7		Task 7 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 7 interrupt overflow flag.
5	INT6		Task 6 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 6 interrupt overflow flag.
4	INT5		Task 5 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 5 interrupt overflow flag.
3	INT4		Task 4 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 4 interrupt overflow flag.
2	INT3		Task 3 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 3 interrupt overflow flag.
1	INT2		Task 2 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 2 interrupt overflow flag.
0	INT1		Task 1 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 1 interrupt overflow flag.

⁽¹⁾ This register is protected by EALLOW and the code security module.

4.4 Execution Registers

The CLA program counter is initialized by the appropriate MVECTx register when an interrupt is received and a task begins execution. The MPC points to the instruction in the decode 2 (D2) stage of the CLA pipeline. After a MSTOP operation, if no other tasks are pending, the MPC will remain pointing to the MSTOP instruction. The MPC register can be read by the main C28x CPU for debug purposes. The main CPU cannot write to MPC.

Register Set

4.4.1 MPC Register

The MPC register is described in Figure 13 and described in Table 13.

Figure 13. Program Counter (MPC)

15	12	11	0
Re	eserved	MPC	
	R-0	R-0	

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

Table 13. Program Counter (MPC) Field Descriptions

Bits	Name	Value	Description ⁽¹⁾
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.
11-0	MPC	0000 - 0FFF	Points to the instruction currently in the decode 2 phase of the CLA pipeline. The value is the offset from the first address in the CLA program space.

⁽¹⁾ This register is protected by the code security module. The main CPU can read this register for debug purposes but it can not write to it.

4.4.2 MSTF Register

The CLA status register (MSTF) reflects the results of different operations. These are the basic rules for the flags:

- Zero and negative flags are cleared or set based on:
 - floating-point moves to registers
 - the result of compare, minimum, maximum, negative and absolute value operations
 - the integer result of operations such as MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32
- Overflow and underflow flags are set by floating-point 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.

The MSTF register is shown in Figure 14 and described in Table 14.

						00/0	Juluo	1.09.010	. (.,				
31						24	23							16
			Reserved							RI	PC			
R/W-0										R/\	N-0			
15		12	11	10	9	8	7	6	5	4	3	2	1	0
	RPC		MEALLOW	Reserved	RND32	Res	erved	TF	Res	erved	ZF	NF	LUF	LVF
	R/W-0		R/W-0	R-0	R/W-0	R	-0	R/W-0	R	-0	R/W-0	R/W-0	R/W-0	R/W-0

Figure 14. CLA Status Register (MSTF)

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



Bits	Field	Value	Description ⁽¹⁾
31 - 24	Reserved	0	Reserved for future use
23 - 12	RPC		Return program counter.
			The RPC is used to save and restore the MPC address by the MCCNDD and MRCNDD operations.
11	MEALLOW		This bit enables and disables CLA write access to EALLOW protected registers. This is independent of the state of the EALLOW bit in the main CPU status register. This status bit can be saved and restored by the MMOV32 STF instruction.
		0	The CLA cannot write to EALLOW protected registers. This bit is cleared by the MEDIS CLA instruction.
		1	The CLA is allowed to write to EALLOW protected registers. This bit is set by the MEALLOW CLA instruction.
10	Reserved	0	Any writes to these bit(s) must always have a value of 0.
9	RND32		Round 32-bit Floating-Point Mode
			Use the MSETFLG and MMOV32 MSTF instructions to change the rounding mode.
		0	If this bit is zero, the MMPYF32, MADDF32 and MSUBF32 instructions will round to zero (truncate).
		1	If this bit is one, the MMPYF32, MADDF32 and MSUBF32 instructions will round to the nearest even value.
8 - 7	Reserved	0	Reserved for future use
6	TF		Test Flag
			The TESTTF instruction can modify this flag based on the condition tested. The MSETFLG and MMOV32 MSTF, mem32 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 - 4	Reserved		These two bits may change based on integer results. These flags are not, however, used by the CLA and therefore marked as reserved.
3	ZF		Zero Flag ^{(2) (3)}
			 Instructions that modify this flag based on the floating-point value stored in the destination register: MMOV32, MMOVD32, MOVDD32, ABSF32, MNEGF32
			 Instructions that modify this flag based on the floating-point result of the operation: MCMPF32, MMAXF32, and MMINF32
			 Instructions that modify this flag based on the integer result of the operation: MMOV16_MAND32_MOR32_MXOR32_MCMP32_MASR32_MLSR32 and MLSL32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag
		0	The value is not zero.
		1	The value is zero.
2	NF		Negative Flag ^{(2) (3)}
			 Instructions that modify this flag based on the fleating point value stored in the destination register;
			MMOV32 MMOVD32 MOVDD32 ABSE32 MNEGE32
			 Instructions that modify this flag based on the floating-point result of the operation: MCMPE32_MMAXE32_and MMINE32
			 Instructions that modify this flag based on the integer result of the operation:
			MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32 and MLSL32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	The value is not negative.
		1	The value is negative.

⁽¹⁾ This register is protected by the code security module. The main CPU can read this register for debug purposes but it can not write to it.

⁽²⁾ A negative zero floating-point value is treated as a positive zero value when configuring the ZF and NF flags.

⁽³⁾ A DeNorm floating-point value is treated as a positive zero value when configuring the ZF and NF flags.



Bits	Field	Value	Description ⁽¹⁾
1	LUF		Latched Underflow Flag
			The following instructions will set this flag to 1 if an underflow occurs: MMPYF32, MADDF32, MSUBF32, MMACF32, MEINVF32, MEISQRTF32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	An underflow condition has not been latched.
		1	An underflow condition has been latched.
0	LVF		Latched Overflow Flag
			The following instructions will set this flag to 1 if an overflow occurs: MMPYF32, MADDF32, MSUBF32, MMACF32, MEINVF32, MEISQRTF32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	An overflow condition has not been latched.
		1	An overflow condition has been latched.

TEXAS INSTRUMENTS

www.ti.com

Pipeline

5 Pipeline

This section describes the CLA pipeline stages and presents cases where pipeline alignment must be considered.

5.1 Pipeline Overview

The CLA pipeline is very similar to the C28x pipeline. The pipeline has eight stages:

• Fetch 1 (F1)

During the F1 stage the program read address is placed on the CLA program address bus.

• Fetch 2 (F2)

During the F2 stage the instruction is read using the CLA program data bus.

Decode 1 (D1)

During D1 the instruction is decoded.

• Decode 2 (D2)

Generate the data read address. Changes to MAR0 and MAR1 due to post-increment using indirect addressing takes place in the D2 phase. Conditional branch decisions are also made at this stage based on the MSTF register flags.

• Read 1 (R1)

Place the data read address on the CLA data-read address bus. If a memory conflict exists, the R1 stage will be stalled.

Read 2 (R2)

Read the data value using the CLA data read data bus.

• Execute (EXE)

Execute the operation. Changes to MAR0 and MAR1 due to loading an immediate value or value from memory take place in this stage.

Write (W)

Place the write address and write data on the CLA write data bus. If a memory conflict exists, the W stage will be stalled.

5.2 CLA Pipeline Alignment

The majority of the CLA instructions do not require any special pipeline considerations. This section lists the few operations that do require special consideration.

Write Followed by Read

In both the CLA pipeline the read operation occurs before the write. This means that if a read operation immediately follows a write, then the read will complete first as shown in Table 15. In most cases this does not cause a problem since the contents of one memory location does not depend on the state of another. For accesses to peripherals where a write to one location can affect the value in another location the code must wait for the write to complete before issuing the read as shown in Table 16. This behavior is different for the 28x CPU. For the 28x CPU any write followed by read to the same location is protected by what is called write-followed-by-read protection. This protection automatically stalls the pipeline so that the write will complete before the read. In addition some peripheral frames are protected such that a 28x CPU write to one location within the frame will always complete before a read to the frame. The CLA does not have this protection mechanism. Instead the code must wait to perform the read.

TEXAS INSTRUMENTS

www.ti.com

Pipeline

Instruction	F1	F2	D1	D2	R1	R2	E	w
I1 MMOV16 @Reg1, MR3	l1							
I2 MMOV16 MR2, @Reg2	12	11						
		12	l1					
			12	I1				
				12	11			
					12	l1		
						12	11	
							12	11

			-					
Instruction	F1	F2	D1	D2	R1	R2	Е	w
I1 MMOV16 @Reg1, MR3	l1							
12	12	11						
13	13	12	11					
14	14	13	12	l1				
I5 MMOV16 MR2, @Reg2	15	14	13	12	l1			
		15	14	13	12	11		
			15	14	13	12	11	
				15	14	13	12	l1
					15	14	13	
						15	14	
							15	

Table 16. Write Followed by Read - Write Occurs First

Delayed Conditional instructions: MBCNDD, MCCNDD and MRCNDD

Referring to Example 1, the following applies to delayed conditional instructions:

– **I1**

11 is the last instruction that can effect the CNDF flags for the branch, call or return instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MBCNDD, MCCNDD or MRCNDD is in the D2 phase.

- I2, I3 and I4

The three instructions proceeding MBCNDD can change MSTF flags but will have no effect on whether the MBCNDD instruction branches or not. This is because the flag modification will occur after the D2 phase of the branch, call or return instruction. These three instructions must not be a MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

- I5, I6 and I7

The three instructions following a branch, call or return are always executed irrespective of whether the condition is true or not. These instructions must not be MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

For a more detailed description refer to the functional description for MBCNDD, MCCNDD and MRCNDD.



Pipeline

Example 1. Code Fragment For MBCNDD, MCCNDD or MRCNDD

<Instruction 1> ; Il Last instruction that can affect flags for ; the branch, call or return operation <Instruction 2> ; I2 Cannot be stop, branch, call or return ; I3 Cannot be stop, branch, call or return <Instruction 4> ; I4 Cannot be stop, branch, call or return <branch/call/ret> ; MBCNDD, MCCNDD or MRCNDD ; I5-I7: Three instructions after are always ; executed whether the branch/call or return is ; taken or not <Instruction 5> ; I5 Cannot be stop, branch, call or return <Instruction 7> ; I6 Cannot be stop, branch, call or return <Instruction 8> ; I8 <Instruction 9> ; I9

Stop or Halting a Task: MSTOP and MDEBUGSTOP

The MSTOP and MDEBUGSTOP instructions cannot be placed three instructions before or after a conditional branch, call or return instruction (MBCNDD, MCCNDD or MRCNDD). Refer to Example 1. To single-step through a branch/call or return, insert the MDEBUGSTOP at least four instructions back and step from there.

Loading MAR0 or MAR1

A load of auxiliary register MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Referring to Example 2, the following applies when loading the auxiliary registers:

– I1 and I2

The two instructions following the load instruction will use the value in MAR0 or MAR1 before the update occurs.

– I3

Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win and the auxiliary register will not be updated with #_X.

- 14

Starting with the 4th instruction MAR0 or MAR1 will have the new value.

Example 2. Code Fragment for Loading MAR0 or MAR1

```
; Assume MAR0 is 50 and #_X is 20
MMOVI16 MAR0, #_X ; Load MAR0 with address of X (20)
<Instruction 1> ; I1 Will use the old value of MAR0 (50)
<Instruction 2> ; I2 Will use the old value of MAR0 (50)
<Instruction 3> ; I3 Cannot use MAR0
<Instruction 4> ; I4 Will use the new value of MAR0 (20)
<Instruction 5> ; I5 Will use the new value of MAR0 (20)
....
```

5.2.1 ADC Early Interrupt to CLA Response

The 2803x ADC offers the option to generate an early interrupt pulse when the ADC begins conversion.

This option is selected by setting the ADCCTL1[INTPULSEPOS] bit as documented in the TMS320x2802x, x2803x Piccolo Analog-to-Digital Converter and Comparator Reference Guide (<u>SPRUGE5</u>). If this option is used to start a CLA task then the CLA will be able to read the result as soon as the conversion completes and the ADC result register updates. This just-in-time sampling along with the low interrupt response of the CLA enable faster system response and higher frequency control loops.

The timing for the ADC conversion is shown in the ADC Reference Guide timing diagrams. From a CLA perspective, the pipeline activity is shown in Table 17. The 8th instruction is in the R2 phase just in time to read the result register. While the first 7 instructions in the task (I1 to I7) will enter the R2 phase of the pipeline too soon to read the conversion, they can be efficiently used for pre-processing calculations needed by the task.

ADC Activity	CLA Activity	F1	F2	D1	D2	R1	R2	Е	W
Sample									
Sample									
Sample									
Conversion (1)	Interrupt Received								
Conversion (2)	Task Startup								
Conversion (3)	Task Startup								
Conversion (4)	11	11							
Conversion (5)	12	12	l1						
Conversion (6)	13	13	12	l1					
Conversion (7)	14	14	13	12	11				
Conversion (8)	15	15	14	13	12	l1			
Conversion (9)	16	16	15	14	13	12	11		
Conversion (10)	17	17	16	15	14	13	12		
Conversion (11)	18 Read ADC RESULT	18	17	16	15	14	13		
Conversion (12)			18	17	16	15	14		
Conversion (13)				18	17	16	15		
Conversion Complete					18	17	16		
RESULT Latched						18	17		
RESULT Available							18		

Table 17. ADC to CLA Early Interrupt Response

5.3 Parallel Instructions

Parallel instructions are single opcodes that perform two operations in parallel. The following types of parallel instructions are available: math operation in parallel with a move operation, or two math operations in parallel. Both operations complete in a single cycle and there are no special pipeline alignment requirements.

Example 3. Math Operation with Parallel Load

```
; MADDF32 || MMOV32 instruction: 32-bit floating-point add with parallel move
; MADDF32 is a 1 cycle operation
    MADDF32 MR0, MR1, #2 ; MR0 = MR1 + 2,
    || MMOV32 MR1, @Val ; MR1 gets the contents of Val
    ; <-- MMOV32 completes here (MR1 is valid)
    ; <-- DDF32 completes here (MR0 is valid)
    ; Any instruction, can use MR1 and/or MR0
```

Example 4. Multiply with Parallel Add

SPRUGE6B-May 2009-Revised May 2010



Pipeline

www.ti.com

Example 4. Multiply with Parallel Add (continued)

;	MMPYF32 MADDF32 instruction	32-bit floating-point multiply with parallel add
;	MMPYF32 is a 1 cycle operation	
;	MADDF32 is a 1 cycle operation	
	MMPYF32 MRO, MR1, MR3	; MRO = MR1 * MR3
	MADDF32 MR1, MR2, MR0	; MR1 = MR2 + MR0 (Uses value of MR0 before MMPYF32)
		; < MMPYF32 and MADDF32 complete here (MR0 and MR1 are valid)
	MMPYF32 MR1, MR1, MR0	; Any instruction, can use MR1 and/or MR0


6 Instruction Set

This section describes the assembly language instructions of the control law accellerator. Also described are parallel operations, conditional operations, resource constraints, and addressing modes. The instructions listed here are independent from C28x and C28x+FPU instruction sets.

6.1 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. CLA 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 C28x CLA are given in Table 18.

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
#SHIFT	Immediate value of 1 to 32 used for arithmetic and logical shifts.
addr	Opcode field indicating the addressing mode
CNDF	Condition to test the flags in the MSTF register
FLAG	Selected flags from MSTF register (OR) 8 bit mask indicating which floating-point status flags to change
MAR0	auxiliary register 0
MAR1	auxiliary register 1
MARx	Either MAR0 or MAR1
mem16	16-bit memory location accessed using direct or indirect addressing modes
mem32	32-bit memory location accessed using direct or indirect addressing modes
MRa	MR0 to MR3 registers
MRb	MR0 to MR3 registers
MRc	MR0 to MR3 registers
MRd	MR0 to MR3 registers
MRe	MR0 to MR3 registers
MRf	MR0 to MR3 registers
MSTF	CLA Floating-point Status Register
shift	Opcode field indicating the number of bits to shift.
VALUE	Flag value of 0 or 1 for selected flag (OR) 8 bit mask indicating the flag value; 0 or 1

Table 18. Operand Nomenclature



Instruction Set

Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

	Description
	Description
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
Opcode	This section shows the opcode for the instruction
Description	Detailed description of the instruction execution is described. Any constraints on the operands imposed by the processor or the assembler are discussed.
Restrictions	Any constraints on the operands or use of the instruction imposed by the processor are discussed.
Pipeline	This section describes the instruction in terms of pipeline cycles as described in Section 5
Example	Examples of instruction execution. If applicable, register and memory values are given before and after instruction execution. Some examples are code fragments while other examples are full tasks that assume the CLA is correctly configured and the main CPU has passed it data.
Operands	Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

Table 19. INSTRUCTION dest, source1, source2 Short Description



6.2 Addressing Modes and Encoding

The CLA uses the same address to access data and registers as the main CPU. For example if the main CPU accesses an ePWM register at address 0x00 6800, then the CLA will access it using address 0x6800. Since all CLA accessible memory and registers are within the low 64k x 16 of memory, only the low 16-bits of the address are used by the CLA.

To address the CLA data memory, message RAMs and shared peripherals, the CLA supports two addressing modes:

- Direct addressing mode: Uses the address of the variable or register directly.
- Indirect addressing with 16-bit post increment. This mode uses either XAR0 or XAR1.

The CLA does not use a data page pointer or a stack pointer. The two addressing modes are encoded as shown in Table 20.

Addressing Mode	'addr' Opcode Field Encode ⁽¹⁾	Description			
@dir	0000	Direct Addressing Mode	•		
		Example 1: MMOV32 MR1, @_VarA			
		Example 2: MMOV32 MR	1, @_EPwm1Regs.CMPA.all		
		In this case the 'mmmm mmmm mmmm mmmm' opcode field will be populated with the 16-bit address of the variable. This is the low 16-bits of the address that you would use to access the variable using the main CPU.			
		For example @_VarA will populate the address of the variable VarA. and @_EPwm1Regs.CMPA.all will populate the address of the CMPA register.			
*MAR0[#imm16]++	0001	MAR0 Indirect Addressing with 16-bit Immediate Post Increment			
*MAR1[#imm16]++	0010	MAR1 Indirect Addressing with 16-bit Immediate Post Increment			
		addr = MAR0 (or MAR1) MAR0 (or MAR1) += #imm16	Access memory using the address stored in MAR0 (or MAR1). Then post increment MAR0 (or MAR1) by #imm16.		
		Example 1: MMOV32 MR	0, *MAR0[2]++		
		Example 2: MMOV32 MR1, *MAR1[-2]++			
		For a post increment of 0 the assembler will accept both *MAR0 and *MAR0[0]++.			
		The 'mmmm mmmm mmmm' opcode field will be populated with the signed 16-bit pointer offset. For example if #imm16 is 2, then the opcode field will be 0x0002. Likewise if #imm16 is -2, then the opcode field will be 0xFFFE.			
		If addition of the 16-bit im 16-bit boundary.	mediate causes overflow, then the value will wrap around on a		

Table 20. Addressing Modes

⁽¹⁾ Values not shown are reserved.

Encoding for the shift fields in the MASR32, MLSR32 and MLSL32 instructions is shown in Table 21

Table 21. Shift Field Encoding

Shift Value	'shift' Opcode Field Encode
1	0000
2	0001
3	0010
32	1111

Table 22 shows the condition field encoding for conditional instructions such as MNEGF, MSWAPF, MBCNDD, MCCNDD and MRCNDD



www.ti.com Table 22. Condition Field Encoding Description MSTF Flags Tested

Encode (1)	CNDF	Description	MSTF 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 OR 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 ⁽²⁾	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 and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.



6.3 Instructions

The instructions are listed alphabetically, preceded by a summary. Table 23. Instructions

Title

Page

MABSF32 MRa, MRb — 32-bit Floating-Point Absolute Value	43
MADD32 MRa, MRb, MRc — 32-bit Integer Add	44
MADDF32 MRa, #16FHi, MRb — 32-bit Floating-Point Addition	45
MADDF32 MRa, MRb, MRc — 32-bit Floating-Point Addition	48
MADDF32 MRd, MRe, MRf MMOV32 mem32, MRa — 32-bit Floating-Point Addition with Parallel Move	49
MADDF32 MRd, MRe, MRf MMOV32 MRa, mem32 — 32-bit Floating-Point Addition with Parallel Move	50
MAND32 MRa, MRb, MRc — Bitwise AND	52
MASR32 MRa, #SHIFT — Arithmetic Shift Right	53
MBCNDD 16BitDest {, CNDF} — Branch Conditional Delayed	54
MCCNDD 16BitDest {, CNDF} — Call Conditional Delayed	59
MCMP32 MRa, MRb — 32-bit Integer Compare for Equal, Less Than or Greater Than	63
MCMPF32 MRa. MRb — 32-bit Floating-Point Compare for Equal. Less Than or Greater Than	64
MCMPF32 MRa. #16FHi — 32-bit Floating-Point Compare for Equal. Less Than or Greater Than	65
MDEBUGSTOP — Debug Stop Task	67
MEALLOW — Enable CLA Write Access to EALLOW Protected Registers	68
MEDIS — Disable CLA Write Access to FALLOW Protected Registers	69
MEINVE32 MRa MRb — 32-bit Floating-Point Reciprocal Approximation	70
MEISORTE32 MRa MRb — 32-bit Floating-Point Square-Root Reciprocal Approximation	72
ME32TOI16 MRa MRb — Convert 32-bit Floating-Point Value to 16-bit Integer	74
ME32TOI16R MRa MRb — Convert 32-bit Floating-Point Value to 16-bit Integer and Round	75
ME32TOI32 MRa MRb — Convert 32-bit Floating-Point Value to 32-bit Integer and Round	76
MF32TOLU16 MRa MRb — Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer	77
ME32TOUI16P MPa MPb — Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer and Round	78
ME32TOUTOR MRA, MRD — Convert 32-bit Floating Point Value to 16 bit Unsigned Integer	70
MEDACE22 MRa, MRb — Convert 52-bit Floating-Foint Value to To-bit Onsigned Integer	20
MISTOF32 MRa, MRb — Flactional Folition of a 32-bit Floating Point Value	00
MICTOF32 MRa, MRD — Convert 10-bit Integer to 32-bit Floating-Point Value	01
MICOLOF32 MRa, mem 10 — Convert 10-bit Integer to 32-bit Floating-Point Value	02
MISZTOF32 MRa, mem32 — Convert 32-bit Integer to 32-bit Floating-Point Value	83
MISZIOF32 MRa, MRD — Convert 32-bit Integer to 32-bit Floating-Point Value	84
MLSL32 MRA, #SHIFT — Logical Shift Left	85
MLSR32 MRa, #SHIFI — Logical Shift Right	86
MMACF32 MR3, MR2, MRd, MRe, MRf MMOV32 MRa, mem32 — 32-bit Floating-Point Multiply and Accumulate with Parallel Move	। 87
MMAXE32 MRa. MRb — 32-bit Eloating-Point Maximum	90
MMAXF32 MRa #16FHi — 32-bit Floating-Point Maximum	Q1
MMINE32 MRa MRb32-bit Eloating-Point Minimum	02
MMINE32 MRa, #16EHi — 32-bit Floating-Point Minimum	02
MMOV16 MARy mem16 — Load MAR1 with 16-bit Value	07
MMOV16 man16 MARy — Move 16-bit Auxiliary Register Contents to Memory	00
MMOV16 mem16, MPa — Move 16-bit Floating-Doint Register Contents to Memory	100
MMOV22 mom22 MPa Move 22 bit Floating Point Register Contents to Memory	100
MMOV32 mem32, MRTE — Move 32-bit MSTE Register to Momony	101
MMOV22 MP2 mom22 (CNDE) Conditional 22 bit Mova	102
MMOV22 MDa MDb (CNDE) Conditional 22 bit Move	103
MMOV/22 MSTE mem22 May 22 bit Value from Memory to the MOTE Deviator	100
MMOVD22 MDa mem32 - Move 32 bit Value from Memory to the MDIF Kegister	107
WINOVDSZ WIKA, MEMSZ — WOVE SZ-DIL VALUE NOM WEMORY WITH DATA COPY	108

Table 23. Instructions (continued)

MMOVIZ MRa, #16FHi — Load the Upper 16-bits of a 32-bit Floating-Point Register	112
MMOVZ16 MRa, mem16 — Load MRx with 16-bit Value	113
MMPYF32 MRa, MRb, MRc — 32-bit Floating-Point Multiply	115
MMPYF32 MRa, #16FHi, MRb — 32-bit Floating-Point Multiply	116
MMPYF32 MRa, MRb, #16FHi — 32-bit Floating-Point Multiply	118
MMPYF32 MRa, MRb, MRc MADDF32 MRd, MRe, MRf — 32-bit Floating-Point Multiply with Parallel Add	120
MMPYF32 MRd, MRe, MRf MMOV32 MRa, mem32 — 32-bit Floating-Point Multiply with Parallel Move	122
MMPYF32 MRd, MRe, MRf MMOV32 mem32, MRa — 32-bit Floating-Point Multiply with Parallel Move	124
MMPYF32 MRa, MRb, MRc MSUBF32 MRd, MRe, MRf — 32-bit Floating-Point Multiply with Parallel Subtract	125
MNEGF32 MRa, MRb{, CNDF} — Conditional Negation	126
MNOP — No Operation	128
MOR32 MRa, MRb, MRc — Bitwise OR	129
MRCNDD {CNDF} — Return Conditional Delayed	130
MSETFLG FLAG, VALUE —Set or clear selected floating-point status flags	133
MSTOP — Stop Task	134
MSUB32 MRa, MRb, MRc — 32-bit Integer Subtraction	136
MSUBF32 MRa, MRb, MRc — 32-bit Floating-Point Subtraction	137
MSUBF32 MRa, #16FHi, MRb — 32-bit Floating Point Subtraction	138
MSUBF32 MRd, MRe, MRf MMOV32 MRa, mem32 — 32-bit Floating-Point Subtraction with Parallel Move	139
MSUBF32 MRd, MRe, MRf MMOV32 mem32, MRa — 32-bit Floating-Point Subtraction with Parallel Move	141
MSWAPF MRa, MRb {, CNDF} — Conditional Swap	142
MTESTTF CNDF — Test MSTF Register Flag Condition	144
MUI16TOF32 MRa, mem16 — Convert unsigned 16-bit integer to 32-bit floating-point value	146
MUI16TOF32 MRa, MRb — Convert unsigned 16-bit integer to 32-bit floating-point value	147
MUI32TOF32 MRa, mem32 — Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value	148
MUI32TOF32 MRa, MRb — Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value	149
MXOR32 MRa, MRb, MRc — Bitwise Exclusive Or	150



MABSF32 MRa, MRb 32-bit Floating-Point Absolute Value

Operands

	MRa	CL	CLA floating-point destination register (MR0 to MR3)					
	MRb	CL	A floating-point so	urce register (MR0 t	o MR3)			
Opcode	LSW: 0000 MSW: 0111	LSW: 0000 0000 bbaa MSW: 0111 1110 0010 0000						
Description	The abso modified I	lute value of by the MAB	of MRb is loade SF32 instructio	d into MRa. Onl on.	y the sign bit of t	the operand is		
	e	else {MRa =	MRb};					
Flags	This instru	uction modi	fies the following	ng flags in the M	ISTF register:			
	Flag	TF	ZF	NF	LUF	LVF	_	
	Modified	No	Yes	Yes	No	No	_	
	The MST NF = 0; ZF = 0; if (MRa(<pre>The MSTF register flags are modified as follows: NF = 0; ZF = 0; if (MRa(30:23) == 0) ZF = 1;</pre>						
Pipeline	This is a s	This is a single-cycle instruction.						
Example	MMOVIZ MR MABSF32 M	10, #-2.0 ; IRO, MRO ;	MR0 = -2.0 (0 MR0 = 2.0 (0x	xC0000000) :40000000), ZF =	= NF = 0			
	MMOVIZ MR MABSF32 M	MMOVIZ MR0, #5.0 ; MR0 = 5.0 (0x40A00000) MABSF32 MR0, MR0 ; MR0 = 5.0 (0x40A00000), ZF = NF = 0						
	MMOVIZ MR MABSF32 M	MMOVIZ MR0, #0.0 ; MR0 = 0.0 MABSF32 MR0, MR0 ; MR0 = 0.0 ZF = 1, NF = 0						
See also	MNEGF3	2 MRa, MR	b {, CNDF}					



Instruction Set

MADD32 MRa, MRb, MRc 32-bit Integer Add

Operands

operanae									
	MRa	MRa CLA floating-point destination register (MR0 to MR3)							
	MRb	MRb CLA floating-point destination register (MR0 to MR3)							
	MRc	CL	A floating-point de	stination register (M	R0 to MR3)				
Opcode	LSW: 0000 000cc bbaa MSW: 0111 1110 1100 0000								
Description	32-bit inte								
	MARa(31:0) = MARb(3)	1:0) + MRc(31:	0);					
Flags	This instru	uction modi	fies the followir	ng flags in the M	ISTF register:				
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	Yes	Yes	No	No			
Pipeline	NF = MRa(ZF = 0; if(MRa(31 This is a s	31); :0) == 0) single-cycle	{ ZF = 1; };						
Example	; Given A ; B ; C ; ; Calcula ; _ClalTask MM MM MM MM MM MM MM MA MA MA	A = (int32): B = (int32): C = (int32): A	1 2 -7 + B + C @_A ; MR0 @_B ; MR1 @_C ; MR2 MR0, MR1 ; A + MR2, MR3 ; A + MR3 ; Sto ; end	= 1 (0x000000 = 2 (0x000000 = -7 (0xFFFFF B B + C = -4 (0 re y2 of task	01) 02) FF9) ×FFFFFFFC)				
See also	MAND32 MASR32 MLSL32 M MLSR32 MOR32 M MXOR32 MSUB32	MRa, MRb MRa, #SHI MRa, #SHIF MRa, #SHI MRa, MRb, MRa, MRb MRa, MRb	, MRc FT T FT MRc , MRc , MRc						



MADDF32 MRa, #16FHi, MRb 32-bit Floating-Point Addition

Operands							
	MRa	CI	_A floating-point de	stination register (M	R0 to MR3)		
	#16FHi 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.						
	MRb	CI	_A floating-point so	urce register (MR0 t	o MR3)		
Opcode	LSW: IIII IIII IIII MSW: 0111 0111 1100 bbaa						
Description	Add MRb t result of th	o the floa e addition	ting-point value ⊨in MRa.	represented by	the immediate o	perand. Store the	
	#16FHi is a floating-po most usefu Some exar (0xBFC000 That is, the MRa = MRb This instruc	a 16-bit in int value. Il for repre nples are 000). The value -1 + #16FH1: ction can	The low 16-bits esenting consta 2.0 (0x400000 assembler will .5 can be repre 0; also be written	that represents of the mantissa nts where the lo 00), 4.0 (0x4080 accept either a l sented as #-1.5 as MADDF32 M	the upper 16-bits a are assumed to west 16-bits of th 00000), 0.5 (0x3F nex or float as the or #0xBFC0. IRa, MRb, #16FF	s of an IEEE 32-bit be all 0. #16FHi is ne mantissa are 0. ² 000000), and -1.5 e immediate value.	
Flags	This instrue	ction mod	ifies the following	ng flags in the M	ISTF register:		
	Flag	TF	ZF	NF	LUF	LVF	
	The MSTF register flags are modified as follows:						
	 LUF = 1 if MADDF32 generates an underflow condition. LVF = 1 if MADDF32 generates an overflow condition. 						
Pipeline	This is a single-cycle instruction.						
Example	; Add to M ; Store th MADDF32	Rl the va e result MR0, #2.0	lue 2.0 in 32- in MRO), MR1 ; MRC	bit floating-po = 2.0 + MR1	oint format		
	<pre>; Add to MR3 the value -2.5 in 32-bit floating-point format ; Store the result in MR2 MADDF32 MR2, #-2.5, MR3 ; MR2 = -2.5 + MR3</pre>						
	; Add to M ; Store th MADDF32	R3 the va e result MR3, #0x3	llue 0x3FC00000 in MR3 SFC0, MR3 ; MR3	(1.5) = 1.5 + MR3			
See also	MADDF32 MADDF32 MADDF32 MADDF32 MMPYF32	MRa, MF MRa, MF MRd, MF MRd, MF MRa, MF	Rb, #16FHi Rb, MRc Re, MRf MMO Re, MRf MMO Rb, MRc MAD	V32 MRa, mem V32 mem32, MI DF32 MRd, MR	32 Ra e, MRf		

MADDF32 MRa, MRb, #16FHi

Operands								
	MRa	CLA floating-point	destination register (MR	0 to MR3)				
	MRb	CLA floating-point	source register (MR0 to	MR3)				
	#16FHi	A 16-bit immediate floating-point value	e value that represents the . The low 16-bits of the	ne upper 16-bits of mantissa are assu	an IEEE 32-bit med to be all 0.			
Opcode	LSW: IIII III MSW: 0111 011	I IIII IIII 1 1100 bbaa						
Description	Add MRb to th result of the a	ne floating-point val ddition in MRa.	ue represented by t	he immediate c	perand. Store the			
	#16FHi is a 16 floating-point v most useful fo Some example (0xBFC00000) That is, the va MRa = MRb + #	#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.						
Flags	This instructio	n can also be writte n modifies the follo	en as MADDF32 MF wing flags in the MS	Ra, #16FHi, MR	b.			
1 1490	Flag T	F ZF	NF	LUF	LVF			
	Modified N	lo No	No	Yes	Yes			
Pineline								
Fipeinie	THIS IS A SILLY							
Example 1	; X is an array of 32-bit floating-point values ; Find the maximum value in an array X ; and store it in Result ;							
	_ClalTask1: MMOVI16 MUI16TOF3 MNOP MNOP MMOV32 LOOP	<pre>MAR1,#_X 2 MR0, @_len MR1, *MAR1[2]++</pre>	<pre>; Start address ; Length of the a ; delay for MAR1 ; delay for MAR1 ; MR1 = X0</pre>	array load load				
	MMOV32 MMAXF32 MADDF32 MCMPF32 MNOP MNOP	<pre>MMOV32 MR2, *MAR1[2]++ ; MR2 = next element MMAXF32 MR1, MR2 ; MR1 = MAX(MR1, MR2) MADDF32 MR0, MR0, #-1.0 ; Decrement the counter MCMPF32 MR0 #0.0 ; Set/clear flags for MBCNDD MNOP MNOP</pre>						
	MBCDD LO MBCNDD LO MMOV32 @_ MNOP MNOP MSTOP	OP, NEQ Result, MR1	; Branch if not e ; Always executed ; Always executed ; Always executed ; Always executed ; End of task	equal to zero 1 1 1				



Example 2	; Show the basic operation of MADDF32 ;
	; Add to MR1 the value 2.0 in 32-bit floating-point format
	; Store the result in MR0 MADDF32 MR0, MR1, #2.0 ; MR0 = MR1 + 2.0
	; Add to MR3 the value -2.5 in 32-bit floating-point format
	; Store the result in MR2
	MADDF32 MR2, MR3, #-2.5 ; MR2 = MR3 + (-2.5)
	; Add to MR0 the value 0x3FC00000 (1.5) ; Store the result in MR0
	MADDF32 MR0, MR0, $\#0x3FC0$; MR0 = MR0 + 1.5
See also	MADDF32 MRa, #16FHi, MRb
	MADDE 32 MRd, MRb, MRC
	MADDE32 MRd, MRc, MRT [] MMOV32 mcd, mch32 MADDE32 MRd, MRe, MRT [] MMOV32 mcm32 MRa
	MMDVE22 MPa MPb MPa II MADDE22 MPd MPa MPf
	IVIIVIE I ESZ IVINA, IVIND, IVINU JI IVIADDESZ IVINU, IVINE, IVINI



MADDF32 MRa, MRb, MRc 32-bit Floating-Point Addition

Operands

	MRa CLA floating-point destination register (MR0 to MR3)									
	MRb	MRb CLA floating-point source register (MR0 to MR3)								
	MRc	CL	A floating-point sc	urce register (MR0	to MR3)					
Opcode	LSW: 000 0 MSW: 0111	LSW: 000 000c bbaa MSW: 0111 1100 0010 0000								
Description	Add the co MRa = MRb	Add the contents of MRc to the contents of MRb and load the result into MRa.								
Flags	This instruc	ction mod	ifies the followi	ng flags in the N	/ISTF register:					
	Flag	TF	ZF	NF	LUF	LVF				
	Modified	No	No	No	Yes	Yes				
Pipeline	• LVF = 1 This is a si	l if MADD ngle-cycle	F32 generates	an overflow cor	ndition.					
Example	<pre>; Given MI ; Calculat ; _ClalTask1 MMOV32 MMOV32 MMPYF32 MMOV32 MADDF32 MMOV32</pre>	<pre>I his is a single-cycle instruction. ; Given M1, X1 and B1 are 32-bit floating point numbers ; Calculate Y1 = M1*X1+B1 ;ClalTask1: MMOV32 MR0,@M1 ; Load MR0 with M1 MMOV32 MR1,@X1 ; Load MR1 with X1 MMPYF32 MR1,MR1,MR0 ; Multiply M1*X1 MMOV32 MR0,@B1 ; and in parallel load MR0 with B1 MADDF32 MR1,MR1,MR0 ; Add M*X1 to B1 and store in MR1</pre>								
See also	MSTOP MADDF32 MADDF32 MADDF32 MADDF32 MMPYF32	MMOV32 @Y1,MR1 ; Store the result MSTOP ; end of task MADDF32 MRa, #16FHi, MRb MADDF32 MRa, MRb, #16FHi MADDF32 MRd, MRe, MRf MMOV32 MRa, mem32 MADDF32 MRd, MRe, MRf MMOV32 mem32, MRa MMPYF32 MRa, MRb, MRc MADDF32 MRd, MRe, MRf								



MADDF32 MRd, MRe, MRf||MMOV32 mem32, MRa 32-bit Floating-Point Addition with Parallel Move

Operands										
-	MRd	CLA floating-point	destination register for	r the MADDF32 (MR	0 to MR3)					
	MRe	MRe CLA floating-point source register for the MADDF32 (MR0 to MR3)								
	MRf	MR3)								
	mem32	32-bit memory loca destination of the N	ition accessed using o /MOV32.	direct or indirect addre	essing. This will be the					
	MRa	CLA floating-point	source register for the	MMOV32 (MR0 to N	/IR3)					
Opcode	LSW: mmmm mmm MSW: 0101 ffe	m mmmm mmmm e ddaa addr								
Description	Perform an Mastore the result mem32.	MADDF32 and a MMOV32 in parallel. Add MRf to the contents of MRe and ult in MRd. In parallel move the contents of MRa to the 32-bit location								
	MRd = MRe + M [mem32] = MRa	Rf; ;								
Flags	This instruction	n modifies the follow	ving flags in the N	ISTF register:						
	Flag T	F ZF	NF	LUF	LVF					
	Modified N	o No	No	Yes	Yes					
	 LUF = 1 if LVF = 1 if 	MADDF32 generate	es an underflow co es an overflow cor	ondition. ndition.						
Pipeline	Both MADDF3	32 and MMOV32 co	mplete in a single	e cycle.						
Example	<pre>; Given A, B and C are 32-bit floating-point numbers ; Calculate Y2 = (A * B) ; Y3 = (A * B) + C ; _ClalTask2: MMOV32 MR0, @_A ; Load MR0 with A MMOV32 MR1, @_B ; Load MR1 with B MMPYF32 MR1, MR1, MR0 ; Multiply A*B MMOV32 MR0, @_C ; and in parallel load MR0 with C MADDF32 MR1, MR1, MR0 ; Add (A*B) to C MMOV32 @_Y2, MR1 ; and in parallel store A*B MMOV32 @_Y3, MR1 ; Store the A*B + C MSTOP ; end of task</pre>									
See also	MADDF32 MF MADDF32 MF MADDF32 MF MMPYF32 MF MADDF32 MF	8a, #16FHi, MRb 8a, MRb, #16FHi 8a, MRb, MRc 8a, MRb, MRc MA 8d, MRe, MRf MM	DDF32 MRd, MR OV32 MRa, mem	e, MRf 32						



MADDF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 32-bit Floating-Point Addition with Parallel Move

Operands									
	MRd	CLA float MRd can	ing-point des not be the sa	tination register for me register as MRa	the MADDF32 (MR0 a.	to MR3).			
	MADDF32 (MR0 to M	MR3)							
	MRf	CLA float	ing-point sou	rce register for the	MADDF32 (MR0 to M	MR3)			
	MRa	CLA float MRa can	tination register for me register as MR	the MMOV32 (MR0 d.	to MR3).				
	mem32	32-bit me for the MI	32-bit memory location accessed using direct or indirect addressing. This is the source for the MMOV32.						
Opcode	LSW: mmmm mn MSW: 0001 ff	umm mmmm mmmm Eee ddaa addr							
Description Perform an MADDF32 and a MMOV32 operation in parallel. Add MRf to the of MRe and store the result in MRd. In parallel move the contents of the 32-bit I mem32 to MRa.						Rf to the contents of e 32-bit location			
	MRd = MRe + MRa = [mem32	MRf; 2];							
Restrictions	The destinat MRa and MF	ion register fo Rd cannot be	r the MAD the same	DF32 and the Nregister.	MMOV32 must be	e unique. That is,			
Flags	This instructi	on modifies th	ne followin	g flags in the M	STF register:				
	Flag			NF	LUF				
	Modified	No	Yes	Yes	Yes	Yes			
	The MSTE re	egister flags a	re modifie	d as follows:					
			ionoratos	an underflow co	ndition				
	• LVE - 1 i				dition				
		I MADDF32 g	enerates	an overnow con	allon.				
	The MMOV3	2 Instruction	will set the	e NF and ZF flag	gs as follows:				
	NF = MRa(31)	;							
	ZF = 0; if(MRa(30:23)	$(3) == 0) \{ ZF \}$	= 1; NF :	= 0; };					
	11 (1110) 50 - 20	, 0, [21	1, 11	0, j,					
Pipeline	The MADDF	32 and the M	MOV32 bo	oth complete in	a single cycle.				
Example 1	; Given A, E ; Calculate ; ;	and C are 3 Y1 = A + 4B Y2 = A + C	2-bit floa	ating-point num	bers				
	_ClalTask1:								
	MMOV32 N MMOV32 N	IRO, @A IRI @B	; Load	MR0 with A MR1 with B					
	MMPYF32	MR1, MR1, #4	.0 ; Multi	iply 4 * B					
	MMOV32 M	IR2, @C	and i	in parallel loa	ld C				
	MADDF32 אסריקרות MADDF32	MR3, MR0, MR MR3, MR0 MR	1 ; Add / 2 ; Add /	A + 4B A + C					
	MMOV32 @	<pre>%Y1, MR3</pre>	; and i	in parallel sto	ore A+4B				
	MMOV32 @	¥2, MR3	; store	e A + C MSTOP					
			, ena (JI LABA					



Instruction Set

Example 2	<pre>; Given A, B and C are 32-bit floating-point numbers ; Calculate Y3 = (A + B) ; Y4 = (A + B) * C ; ClalTack2:</pre>
	<pre>Claifiask2' MMOV32 MR0, @A ; Load MR0 with A MMOV32 MR1, @B ; Load MR1 with B MADDF32 MR1, MR1, MR0 ; Add A+B MMOV32 MR0, @C ; and in parallel load MR0 with C MMPYF32 MR1, MR1, MR0 ; Multiply (A+B) by C MMOV32 @Y3, MR1 ; and in parallel store A+B MMOV32 @Y4, MR1 ; Store the (A+B) * C MSTOP ; end of task</pre>
See also	MADDF32 MRa, #16FHi, MRb MADDF32 MRa, MRb, #16FHi MADDF32 MRa, MRb, MRc MADDF32 MRd, MRe, MRf MMOV32 mem32, MRa MMPYF32 MRa, MRb, MRc MADDF32 MRd, MRe, MRf

MAND32 MRa, MRb, MRc Bitwise AND

Operands

eperanae	MRa CLA floating-point destination register (MR0 to MR3)								
	MRb CLA floating-point source register (MR0 to MR3)								
	MRc	CL	A floating-point sou	irce register (MR0 t	o MR3)				
Opcode	LSW: 0000 MSW: 0111	LSW: 0000 000c bbaa MSW: 0111 1100 0110 0000							
Description	Bitwise AN	Bitwise AND of MRb with MRc. MRa(31:0) = MRb(31:0) AND MRc(31:0);							
Flags	This instruc	ction modi	fies the followin	g flags in the M	STF register:				
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	Yes	Yes	No	No			
	<pre>NF = MRa(3 ZF = 0; if(MRa(31:</pre>	1); 0) == 0)	{ ZF = 1; }						
Pipeline	This is a si	ngle-cycle	instruction.						
Example	MMOVIZ M MMOVXI M	R0, #0 R0, #0	x5555 ; MR0 = xAAAA	0x5555AAAA					
	MMOVIZ M MMOVXI M	R1, #0 R1, #0	x5432 ; MR1 = xFEDC	0x5432FEDC					
	<pre>; 0101 AND 0101 = 0101 (5) ; 0101 AND 0100 = 0100 (4) ; 0101 AND 0011 = 0001 (1) ; 0101 AND 0010 = 0000 (0) ; 1010 AND 1111 = 1010 (A) ; 1010 AND 1110 = 1010 (A) ; 1010 AND 1101 = 1000 (8) ; 1010 AND 1100 = 1000 (8)</pre>								
	MAND32 MR2	, MR1, MR	0 ; MR3 =	0x5410AA88					
See also	MADD32 M MASR32 M MLSL32 M MLSR32 M MOR32 M MXOR32 M MSUB32 M	/IRa, MRb /IRa, #SHI IRa, #SHII IRa, #SHI Ra, MRb, /IRa, MRb /IRa, MRb	, MRc FT FT MRc , MRc , MRc						

MASR32 MRa, #SHIFT Arithmetic Shift Right

Operands

	MRa CLA floating-point source/destination register (MR0 to MR3)							
	#SHIFT	Nu	mber of bits to shif	t (1 to 32)				
Opcode	LSW: 0000 MSW: 0111	0000 Oshi 1011 0100	ftaa 0000					
Description	Arithmetic to 32. MARa(31:0)	shift right of = Arithme	of MRa by the matter shift(MAR	number of bits ir a(31:0) by #SHI	ndicated. The nu	mber of bits can be 1		
Flags	This instru	ction modif	fies the followir	ng flags in the M	STF register:			
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	Yes	Yes	No	No		
	NF = MRa(3 ZF = 0; if(MRa(31:	31); (0) == 0) {	{ ZF = 1; }		5	·		
Pipeline	I his is a si	ingle-cycle	instruction.					
Example	; Given m2 ; x2 ; b2 ; ; Calculat ; m2 ; x2 ; b2 ; [ClalTask2 MMOV32 MMOV32 MMOV32 MASR32 MASR32 MASR32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32	2 = (int32) 2 = (int32) 2 = (int32) 2 = (int32) 2 = m2/2 2 = m2/2 2 = m2/2 2 = m2/4 2 = b2/8 2: MR0, @_m2 MR1, @_m2 MR2, #1 MR1, #2 MR2, #3 @_m2, MR0 @_x2, MR1 @_b2, MR2 : end of ta	<pre>32 32 64 -128 ; MR0 = 32 (0 ; MR1 = 64 (0 ; MR2 = -128 ; MR0 = 16 (0 ; MR1 = 16 (0 ; MR1 = 16 (0 ; MR2 = -16 (; store resul ask</pre>	x00000020) x0000040) (0xFFFFFF80) x00000010) x00000010) 0xFFFFFFF0) ts				
See also	MADD32 M MAND32 M MLSL32 M MLSR32 M MOR32 M MXOR32 M MSUB32 M	MRa, MRb, MRa, MRb, IRa, #SHIF /IRa, #SHIF Ra, MRb, I MRa, MRb, MRa, MRb,	MRc MRc T T MRc MRc MRc MRc					



MBCNDD 16BitDest {, CNDF} Branch Conditional Delayed

Operands						
	16BitDest	16-bi	t destination	if condition is true		
	CNDF	Optic	onal condition	n tested		
Opcode	LSW: dest MSW: 0111	dest dest d 1001 1000 c	lest Indf			
Description	If the speci MPC value around. Th instruction. address ar	fied condition of therwise erefore a var Since the M e ignored.	on is true, s , continue alue of "0x MPC is onl	then branch by adding the without branching. If the a FFFE" will put the MPC b y 12-bits, unused bits the	e signed 16Bit[address overflo ack to the MB0 upper 4 bits o	Dest value to the ows, it wraps CNDD f the destination
	Please refe	er to the pip	eline section	on for important information	on regarding th	is instruction.
	if (CNDF =	= TRUE) MPC	2 += 16Bit	Dest;		
	CNDF is or	ne of the fol	lowina cor	nditions:		
				Description	MSTF Flags Te	sted
	0000	NEQ		Not equal to zero	ZF == 0	
	0001	EQ		Equal to zero	ZF == 1	
	0010	GT		Greater than zero	ZF == 0 AND N	= == 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 OR 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 ⁽²⁾		Unconditional with flag modification	None	
Restrictions	 (1) Values no (2) This is the be modified (2) The MBCN 	ot shown are re e default opera ed when a con IDD instruct	eserved. ation if no CN iditional opera ion is not a	DF field is specified. This conditation is executed. All other cond	tion will allow the 2 litions will not mod before or after	ZF and NF flags to ify these flags.
	MCCNDD	or MRCND	D instructio	on. Refer to the pipeline so	ection for more	information.
Flags	This instruc	ction does n	ot modify	flags in the MSTF register	r.	
	Flag	TF	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No
Pipeline	The MBCN each brand and three a taken or no depends of effective nu number of	IDD instruct th 6 instruct after the bra bt taken dep n how many umber of cyc cycles for a	ion by itse ion slots a nch instruc- ends on th slots are cles for a b branch ta	If is a single-cycle instruct re executed; three before ction (I5-I7). The total nun ne usage of these slots. T filled with a MNOP as we pranch can, therefore, ran ken may not be the same	tion. As shown the branch ins nber of cycles hat is, the num Il as which slot ge from 1 to 7 as for a branc	in Table 24 for struction (I2-I4) for a branch aber of cycles is are filled. The cycles. The h not taken.



Referring to Table 24 and Table 25, the instructions before and after MBCNDD have the following properties:

• I1

- I1 is the last instruction that can effect the CNDF flags for the MBCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MBCNDD is in the D2 phase.
- There are no restrictions on the type of instruction for I1.
- I2, I3 and I4
 - The three instructions proceeding MBCNDD can change MSTF flags but will have no effect on whether the MBCNDD instruction branches or not. This is because the flag modification will occur after the D2 phase of the MBCNDD instruction.
 - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- 15, 16 and 17
 - The three instructions following MBCNDD are always executed irrespective of whether the branch is taken or not.
 - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

```
<Instruction 1>
                     ; Il Last instruction that can affect flags for
                    ; the MBCNDD operation
<Instruction 2>
                   ; I2 Cannot be stop, branch, call or return
<Instruction 3> ; I3 Cannot be stop, branch, call or return
<Instruction 4> ; I4 Cannot be stop, branch, call or return
MBCNDD _Skip, NEQ ; Branch to Skip if not eqal to zero
                    ; Three instructions after MBCNDD are always
                    ; executed whether the branch is taken or not
<Instruction 5>
                   ; I5 Cannot be stop, branch, call or return
                   ; I7 Cannot be stop, branch, call or return
; I7 Cannot be stop, branch, call or return
; I8
<Instruction 6> ; I6 Cannot be stop, branch, call or return
<Instruction 7>
<Instruction 8>
<Instruction 9>
                   ; 19
. . . .
_Skip:
 <Destination 1> ; d1 Can be any instruction
 <Destination 2> ; d2
 <Destination 3> ; d3
. . . .
. . . .
MSTOP
. . . .
```

Texas Instruments

Instruction Set

Instruction	F1	F2	D1	D2	R1	R2	Е	w
l1	11							
12	12	11						
13	13	12	11					
14	14	13	12	11				
MBCNDD	MBCNDD	14	13	12	l1			
15	15	MBCNDD	14	13	12	l1		
16	16	15	MBCNDD	14	13	12	11	
17	17	16	15	MBCNDD	14	13	12	
18	18	17	16	15	-	14	13	
19	19	18	17	16	15	-	14	
110	110	19	18	17	16	15	-	
		I10	19	18	17	16	15	
			I10	19	18	17	16	
				l10	19	18	17	
					l10	19	18	
						l10	19	
							110	

Table 25. Pipeline Activity For MBCNDD, Branch Taken								
Instruction	F1	F2	D1	D2	R1	R2	Е	w
11	l1							
12	12	11						
13	13	12	11					
14	14	13	12	11				
MBCNDD	MBCNDD	14	13	12	11			
15	15	MBCNDD	14	13	12	l1		
16	16	15	MBCNDD	14	13	12	11	
17	17	16	15	MBCNDD	14	13	12	
d1	d1	17	16	15	-	14	13	
d2	d2	d1	17	16	15	-	14	
d3	d3	d2	d1	17	16	15	-	
		d3	d2	d1	17	16	15	
			d3	d2	d1	17	16	
				d3	d2	d1	17	
					d3	d2	d1	
						d3	d2	
							d3	



www.	.ti.	CO	m

Example 1

; if (State == 0.1) ; PampState = PampState	1	DAMDMA CK
; algo if (State = 0.01)		KAMPMASK
: CoastState - CoastState	Т	COASTMASK
: else	1	COASTMASK
: SteadyState - SteadyStat		
:	-6	STEADTHASK
/ ClalTack1:		
MMOV32 MR0 @State		
$MCMPF32 MR0 \pm 0.1$;	Affects flags for 1st MBCNDD (A)
MNOP	'	Mileeeb Hugb for the means (M)
MNOP		
MNOP		
MBCNDD Skip1, NEO	;	(A) If State != 0.1, go to Skipl
MNOP ; Always executed		(, , , , , , , , , , , , , , , , , , ,
MNOP ; Always executed		
MNOP ; Always executed		
MMOV32 MR1, @RampState	;	Execute if (A) branch not taken
MMOVXI MR2, #RAMPMASK	;	Execute if (A) branch not taken
MOR32 MR1, MR2	;	Execute if (A) branch not taken
MMOV32 @RampState, MR1	;	Execute if (A) branch not taken
MSTOP	;	end of task if (A) branch not taken
Skip1:		
MCMPF32 MR0,#0.01	;	Affects flags for 2nd MBCNDD (B)
MNOP		
MNOP		
MNOP		
MBCNDD Skip2,NEQ	;	(B) If State != 0.01, go to Skip2
MNOP ; Always executed		
MNOP ; Always executed		
MNOP ; Always executed		The mater of (D) have also as the large
MMOV32 MRI, @COAStState	;	Execute II (B) branch not taken
MMOVXI MRZ, #COASIMASK	΄.	Execute II (B) branch not taken
MURSZ MRI, MRZ	΄.	Execute II (B) branch not taken
MMOV32 @COASISIALE, MRI	'	Execute II (B) branch not taken
Skip2:		
MMOV32 MR3 @SteadyState	;	Executed if (B) branch taken
MMOVXI MR2, #STEADYMASK	;	Executed if (B) branch taken
MOR32 MR3, MR2	;	Executed if (B) branch taken
MMOV32 @SteadyState. MR3	;	Executed if (B) branch taken
MSTOP		

Texas Instruments

www.ti.com

Instruction Set		
Example 2	<pre>; This example is the same ; the code is optimized to ; ; if (State == 0.1) ; RampState = RampState ; else if (State == 0.01) ; CoastState = CoastState ; else ; SteadyState = SteadyStat ;</pre>	e as Example 1, except o take advantage of delay slots RAMPMASK COASTMASK :e STEADYMASK
	_ClalTask2: MMOV32 MR0, @State MCMPF32 MR0, #0.1 MCMPF32 MR0, #0.01 MTESTTF EQ MNOP MBCNDD Skip1, NEQ MMOV32 MR1, @RampState MMOVXI MR2, #RAMPMASK MOR32 MR1, MR2 MMOV32 @RampState, MR1 MSTOP	<pre>; Affects flags for 1st MBCNDD (A) ; Check used by 2nd MBCNDD (B) ; Store EQ flag in TF for 2nd MBCNDD (B) ; (A) If State != 0.1, go to Skip1 ; Always executed ; Always executed ; Always executed ; Always executed ; Execute if (A) branch not taken ; end of task if (A) branch not taken</pre>
	Skip1: MMOV32 MR3, @SteadyState MMOVXI MR2, #STEADYMASK MOR32 MR3, MR2 MBCNDD Skip2, NTF MMOV32 MR1, @CoastState MMOVXI MR2, #COASTMASK MOR32 MR1, MR2 MMOV32 @CoastState, MR1 MSTOP Skip2: MMOV32 @SteadyState, MR3 MSTOP	<pre>; (B) if State != .01, go to Skip2 ; Always executed ; Always executed ; Always executed ; Execute if (B) branch not taken ; end of task if (B) branch not taken ; Executed if (B) branch taken</pre>
Can alan		

See also

MCCNDD 16BitDest, CNDF MRCNDD CNDF



MCCNDD 16BitDest {, CNDF} Call Conditional Delayed

Operands

	16BitDest	16-bit desti	ination if condition is true						
	CNDF	Optional co	ondition to be tested						
Opcode	LSW: dest MSW: 0111	dest dest dest 1001 1001 cndf							
Description	If the spec and make continue co around. Th instruction address ar	ified condition is the call by addin ode execution wi herefore a value of . Since the MPC re ignored.	true, then store the return addr g the signed 16BitDest value to thout making the call. If the add of "0xFFFE" will put the MPC bai is only 12 bits, unused bits the	ess in the RPC field of MSTF o the MPC value. Otherwise, dress overflows, it wraps ack to the MCCNDD upper 4 bits of the destination					
	<pre>Please refe if (CNDF = { RPC = MPC += };</pre>	er to the pipeline == TRUE) return address; = 16BitDest;	section for important information	on regarding this instruction.					
	CNDF is one of the following conditions:								
	Encode ⁽³⁾	CNDF	Description	MSTF 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 OR NF == 1					
	1010	TF	Test flag set	TF == 1					
	1011	NTF	Test flag not set	TF == 0					
	1100	LU	Latched underflow	LUF == 1					
		IV	Latched overflow	LVF == 1					
	1101	L •							
	1101 1110	UNC	Unconditional	None					

Restrictions

The MCCNDD instruction is not allowed three instructions before or after a MBCNDD, MCCNDD, or MRCNDD instruction. Refer to the Pipeline section for more details.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No



Instruction Set

Pipeline

The MCCNDD instruction by itself is a single-cycle instruction. As shown in Table 26, for each call 6 instruction slots are executed; three before the call instruction (I2-I4) and three after the call instruction (I5-I7). The total number of cycles for a call taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a call can, therefore, range from 1 to 7 cycles. The number of cycles for a call taken may not be the same as for a call not taken.

Referring to the following code fragment and the pipeline diagrams in Table 26 and Table 27, the instructions before and after MCCNDD have the following properties:

- I1
 - I1 is the last instruction that can effect the CNDF flags for the MCCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MCCNDD is in the D2 phase.
 - There are no restrictions on the type of instruction for I1.
- I2, I3 and I4
 - The three instructions proceeding MCCNDD can change MSTF flags but will have no effect on whether the MCCNDD instruction makes the call or not. This is because the flag modification will occur after the D2 phase of the MCCNDD instruction.
 - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- 15, 16 and 17
 - The three instructions following MBCNDD are always executed irrespective of whether the branch is taken or not.
 - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

TEXAS INSTRUMENTS

www.ti.com

<instruction 1=""></instruction>	; ;	Il Last instruction that can affect flags for the MCCNDD operation
<instruction 2=""></instruction>	;	12 Cannot be stop, branch, call or return
<instruction 3=""></instruction>	;	13 Cannot be stop, branch, call or return
<instruction 4=""></instruction>	;	14 Cannot be stop, branch, call or return
MCCNDD _func, NEQ	;	Call to func if not eqal to zero
	;	Three instructions after MCCNDD are always
	;	executed whether the call is taken or not
<instruction 5=""></instruction>	;	I5 Cannot be stop, branch, call or return
<instruction 6=""></instruction>	;	I6 Cannot be stop, branch, call or return
<instruction 7=""></instruction>	;	17 Cannot be stop, branch, call or return
<instruction 8=""></instruction>	;	18 The address of this instruction is saved
	;	in the RPC field of the MSTF register.
	;	Upon return this value is loaded into MPC
	;	and fetching continues from this point.
<instruction 9=""></instruction>	;	19
· · · · ·		
_func:		
<destination 1=""></destination>	;	dl Can be any instruction
Destauration O.		-10
<destination 2=""></destination>	;	d2
<destination 2=""> <destination 3=""></destination></destination>	; ;	d2 d3 d4 Joch instruction that are offert floor for
<destination 2=""> <destination 3=""> <destination 4=""></destination></destination></destination>	;;;	d2 d3 d4 Last instruction that can affect flags for
<destination 2=""> <destination 3=""> <destination 4=""></destination></destination></destination>	;;;;	d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation
<destination 2=""> <destination 3=""> <destination 4=""> <destination 5=""></destination></destination></destination></destination>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	d2d3d4 Last instruction that can affect flags for the MRCNDD operationd5 Cannot be stop, branch, call or return
<pre><destination 2=""> <destination 3=""> <destination 4=""> </destination></destination></destination></pre> <pre></pre> <pre><</pre>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	 d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return
<pre><destination 2=""> <destination 3=""> <destination 4=""> </destination></destination></destination></pre> <pre></pre> <pre><</pre>	;;;;;;;;;	 d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return
<pre><destination 2=""> <destination 3=""> <destination 4=""> </destination></destination></destination></pre> <pre></pre> <pre><</pre>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional</instruction></pre>
<pre><destination 2=""> <destination 3=""> <destination 4=""> <destination 5=""> <destination 6=""> <destination 7=""> MRCNDD, UNC</destination></destination></destination></destination></destination></destination></pre>	;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MPCNDD are already</instruction></pre>
<destination 2=""> <destination 3=""> <destination 4=""> <destination 5=""> <destination 6=""> <destination 7=""> MRCNDD, UNC</destination></destination></destination></destination></destination></destination>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MRCNDD are always executed whether the return is taken or not</instruction></pre>
<pre><destination 2=""> <destination 3=""> <destination 4=""> <destination 5=""> <destination 6=""> <destination 7=""> MRCNDD, UNC</destination></destination></destination></destination></destination></destination></pre>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MRCNDD are always executed whether the return is taken or not</instruction></pre>
<pre><destination 2=""> <destination 3=""> <destination 4=""> <destination 5=""> <destination 6=""> <destination 7=""> MRCNDD, UNC <destination 8=""></destination></destination></destination></destination></destination></destination></destination></pre>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MRCNDD are always executed whether the return is taken or not d8 Cannot be stop, branch, call or return</instruction></pre>
<pre><destination 2=""> <destination 3=""> <destination 4=""> <destination 5=""> <destination 6=""> <destination 7=""> MRCNDD, UNC </destination></destination></destination></destination></destination></destination></pre>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MRCNDD are always executed whether the return is taken or not d8 Cannot be stop, branch, call or return d9 Cannot be stop, branch, call or return</instruction></pre>
<pre><destination 2=""> <destination 3=""> <destination 4=""> </destination></destination></destination></pre> <pre><destination 5=""> <destination 6=""> <destination 7=""> </destination></destination></destination></pre> <pre>MRCNDD, UNC </pre> <pre><destination 8=""> <destination 9=""> <destination 10=""> </destination></destination></destination></pre>	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MRCNDD are always executed whether the return is taken or not d8 Cannot be stop, branch, call or return d9 Cannot be stop, branch, call or return d10 Cannot be stop, branch, call or return</instruction></pre>
<pre><destination 2=""> <destination 3=""> <destination 4=""> </destination></destination></destination></pre> <pre><destination 5=""> <destination 6=""> <destination 7=""> MRCNDD, UNC </destination></destination></destination></pre> <pre><destination 8=""> <destination 9=""> <destination 10=""> <destination 11=""> </destination></destination></destination></destination></pre>	;;;; ;;; ; ;; ;;;;	<pre>d2 d3 d4 Last instruction that can affect flags for the MRCNDD operation d5 Cannot be stop, branch, call or return d6 Cannot be stop, branch, call or return d7 Cannot be stop, branch, call or return Return to <instruction 8="">, unconditional Three instructions after MRCNDD are always executed whether the return is taken or not d8 Cannot be stop, branch, call or return d9 Cannot be stop, branch, call or return d10 Cannot be stop, branch, call or return d11</instruction></pre>

MSTOP



Instruction Set

Instruction	F1	F2	D1	D2	R1	R2	E	W	
11	11								
12	12	l1							
13	13	12	l1						
14	14	13	12	l1					
MCCNDD	MCCNDD	14	13	12	l1				
15	15	MCCNDD	14	13	12	11			
16	16	15	MCCNDD	14	13	12	l1		
17	17	16	15	MCCNDD	14	13	12		
18	18	17	16	15	-	14	13		
19	19	18	17	16	15	-	14		
I10	110	19	18	17	16	15	-		
etc		I10	19	18	17	16	15		
			l10	19	18	17	16		
				l10	19	18	17		
					I10	19	18		
						l10	19		
							110		

Table 26. Pipeline Activity For MCCNDD, Call Not Taken

	Table 27. Pipeline Activity For MCCNDD, Call Taken							
Instruction	F1	F2	D1	D2	R1	R2	Е	w
11	l1							
12	12	l1						
13	13	12	l1					
14	14	13	12	11				
MCCNDD	MCCNDD	14	13	12	l1			
15	15	MCCNDD	14	13	12	11		
16	16	15	MCCNDD	14	13	12	l1	
17 ⁽¹⁾	17	16	15	MCCNDD	14	13	12	
d1	d1	17	16	15	-	14	13	
d2	d2	d1	17	16	15	-	14	
d3	d3	d2	d1	17	16	15	-	
etc		d3	d2	d1	17	16	15	
			d3	d2	d1	17	16	
				d3	d2	d1	17	
					d3	d2	d1	
						d3	d2	
							d3	

⁽¹⁾ The RPC value in the MSTF register will point to the instruction following I7 (instruction I8).

Example

See also

MBCNDD #16BitDest, CNDF MMOV32 mem32, MSTF MMOV32 MSTF, mem32 MRCNDD CNDF

;



MCMP32 MRa, MRb 32-bit Integer Compare for Equal, Less Than or Greater Than

Operands								
	MRa	CLA floating-point	t source register (MR0 t	to MR3)				
	MRb	CLA floating-point	t source register (MR0 t	to MR3)				
Opcode	LSW: 0000 00 MSW: 0111 11	00 0000 bbaa 11 0010 0000						
Description	Set ZF and N integers. For	IF flags on the resu a floating point com	It of MRa - MRb white the second s	here MRa and M IPF32.	IRb are 32-bit			
Flags	This instruction	on modifies the follo	wing flags in the M	ISTF register:				
	Flag	TF ZF	NF	LUF	LVF			
	Modified	No Yes	Yes	No	No			
Pipeline	This is a sing	le-cycle instruction.						
Pipeline Example	<pre>This is a single-cycle instruction. Figure A = (int32)1 Figure B = (int32)2 Figure B = (int32)-7 Figure B = MR0, @_A ; MR0 = 1 (0x00000001) MM0V32 MR1, @_B ; MR1 = 2 (0x0000002) MM0V32 MR2, @_C ; MR2 = -7 (0xFFFFFF9) MC0M22 MR2 = MR2 = 0 7 T = 1</pre>							
See also	MCMP32 M MCMP32 M MADD32 MR MSUB32 MR	R0, MR1 ; NF = 1, R1, MR0 ; NF = 0, a, MRb, MRc a, MRb, MRc	ZF = 0 ZF = 0					

MCMPF32 MRa, MRb 32-bit Floating-Point Compare for Equal, Less Than or Greater Than

Operands											
	MRa	MRa CLA floating-point source register (MR0 to MR3)									
	MRb	Cl	_A floating-point so	urce register (MR0 t	o MR3)						
Opcode	LSW: 0000 MSW: 0111	0000 0000	bbaa 0000								
Description	Set ZF and as a logica the expone	d NF flags al compare ent. Basica	on the result o e operation. This ally the bigger th	f MRa - MRb. Th s is possible bea he binary numbe	he MCMPF32 in cause of the IEE er, the bigger the	struction is performed E format offsetting e floating-point value.					
	Special ca	ses for inp	outs:								
	 Negative 	/e zero wi	II be treated as	positive zero.							
	 A deno 	rmalized v	value will be tre	ated as positive	zero.						
	 Not-a-N 	Number (N	laN) will be trea	ted as infinity.							
Flags	This instru	ction mod	ifies the followir	ng flags in the M	ISTF register:						
	Flag	TF	ZF	NF	LUF	LVF					
	Modified	No	Yes	Yes	No	No					
	If(MRa == If(MRa > M If(MRa < M	<pre>If MRa == MRb) {ZF=1; NF=0;} If(MRa > MRb) {ZF=0; NF=0;} If(MRa < MRb) {ZF=0; NF=1;}</pre>									
Pipeline	This is a s	ingle-cycle	e instruction.								
Example	; Behavior	; Behavior of ZF and NF flags for different comparisons									
	MMOVIZ MMOVIZ MCMPF3 MCMPF3 MCMPF3	MR1, MR0, 2 MR1, 32 MR1, 32 MR0, 32 MR0,	#-2.0 ; MR1 = #5.0 ; MR0 = MR0 ; ZF = MR1 ; ZF = MR0 ; ZF =	-2.0 (0xC00000 5.0 (0x40A0000 0, NF = 1 0, NF = 0 1, NF = 0	000) 00)						
See also	MCMPF32 MMAXF32 MMAXF32 MMINF32 MMINF32	2 MRa, #10 2 MRa, #10 2 MRa, MR 2 MRa, #16 MRa, MR	6FHi 6FHi Rb FHi b								



MCMPF32 MRa, #16FHi 32-bit Floating-Point Compare for Equal, Less Than or Greater Than

Operands									
	MRa	CLA floati	ng-point sour	ce register (MR0 to	o MR3)		-		
	#16FHi	A 16-bit in floating-po	nmediate valu bint value. The	e that represents t e low 16-bits of the	he upper 16-bits of mantissa are assu	an IEEE 32-bit med to be all 0.			
Opcode	LSW: IIII III MSW: 0111 100	I IIII IIII 00 1100 00aa							
Description	Compare the operand. Set	value in MRa the ZF and N	with the fl IF flags on	oating-point va (MRa - #16FH	lue represented i:0).	by the immediate			
	#16FHi is a 1 floating-point addressing m are 0. Some e -1.5 (0xBFC0 value. That is	6-bit immedia value. The lo ode is most u examples are 0000). The as , -1.5 can be	te value th w 16-bits c useful for co 2.0 (0x400 ssembler w represente	at represents t f the mantissa onstants where 000000), 4.0 (0 /ill accept eithe d as #-1.5 or #	he upper 16-bits are assumed to the lowest 16-b x40800000), 0.3 r a hex or float 0xBFC0.	s of an IEEE 32-bit be all 0. This bits of the mantissa 5 (0x3F000000), ar as the immediate	a nd		
	The MCMPF3 because of th binary numbe	The MCMPF32 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. 								
	Denormali	 Denormalized value will be treated as positive zero. 							
	 Not-a-Nun 	nber (NaN) w	ill be treate	ed as infinity.					
Flags	This instruction	n modifies th	e following	flags in the M	STF register:				
	Flag	F 2	ZF	NF	LUF	LVF			
	Modified N	۱o `	Yes	Yes	No	No			
	The MSTF register flags are modified as follows: If(MRa == #16FHi:0) {ZF=1, NF=0;} If(MRa > #16FHi:0) {ZF=0, NF=0;} If(MRa < #16FHi:0) {ZF=0, NF=1;}								
Pipeline	This is a singl	e-cycle instru	iction						
Example 1	; Behavior of	ZF and NF f	lags for d	lifferent comp	arisons				
	MMOVIZ MMOVIZ MCMPF32 MCMPF32 MCMPF32	MR1, #-2.0 MR0, #5.0 MR1, #-2.2 MR0, #6.5 MR0, #5.0	; MR1 = -; ; MR0 = 5; ; ZF = 0; ; ZF = 0; ; ZF = 1;	2.0 (0xC00000 5.0 (0x40A0000 NF = 0 NF = 1 NF = 0	00) 0)				



		<u> </u>
Inctri	intinn	Sof
111301	$i \cup i \cup i$	$\overline{\mathbf{U}}$

Example 2	<pre>, A is an array of 32-bit floating-point values ; and has len elements. Find the maximum value in ; the array and store it in Result ; ; Note: MCMPF32 and MSWAPF can be replaced with MMAXF32 ; _ClalTask1: MMOVI16 MAR1,#_X ; Start address MUI16TOF32 MR0, @_len ; Length of the array MNOP ; delay for MAR1 load MNOP ; delay for MAR1 load MMOV32 MR1, *MAR1[2]++ ; MR1 = X0</pre>						
	LOOP MMOV32 MR2, *MAR1[2]++ MCMPF32 MR2, MR1 MSWAPF MR1, MR2, GT MADDF32 MR0, MR0, #-1.0 MCMPF32 MR0 #0.0 MNOP MNOP MNOP MBCNDD LOOP, NEQ	<pre>; MR2 = next element ; Compare MR2 with MR1 ; MR1 = MAX(MR1, MR2) ; Decrement the counter ; Set/clear flags for MBCNDD ; Branch if not equal to zero</pre>					
	MMOV32 @_Result, MR1 MNOP MNOP MSTOP	<pre>; Always executed ; Always executed ; Always executed ; End of task</pre>					
See also	MCMPF32 MRa, MRb MMAXF32 MRa, #16FHi MMAXF32 MRa, MRb MMINF32 MRa, #16FHi MMINF32 MRa, MRb						

TEXAS INSTRUMENTS

MDEBUGSTOP	Debug Stop Task							
Operands								
	none	TI	nis instruction does	not have any opera	ands			
Opcode	LSW: 0000 MSW: 0111	0000 0000 1111 0110	0000 0000					
Description Restrictions	When CLA breakpoints are enabled, the MDEBUGSTOP instruction is used to halt a task so that it can be debugged. That is, MDEBUGSTOP is the CLA breakpoint. If CLA breakpoints are not enabled, the MDEBUGSTOP instruction behaves like a MNOP. Unlike the MSTOP, the MIRUN flag is not cleared and an interrupt is not issued. A single-step or run operation will continue execution of the task.							
	MBCNDD	, MCCNDI	D or MRCNDD	instruction.				
Flags	This instru	uction does	s not modify flag	gs in the MSTF	register.			
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No		
Pipeline	This is a s	single-cycle	e instruction.					
Example	;							
See also	MSTOP							



MEALLOW	Enable C	LA Write	Access to EAI	LOW Protecte	d Registers					
Operands										
	none	Tł	nis instruction does	not have any operation	ands					
Opcode	LSW: 0000 MSW: 0111	0000 0000 1111 1001	0000							
Description	This instr set, the C against C	uction sets LA is allow LA writes t	the MEALLOV ved write acces o protected reg	/ bit in the CLA s to EALLOW p jisters, use the I	status register M rotected registers MEDIS instruction	STF. When this bit is s. To again protect n.				
	MEALLO MEALLO the main main CPU CLA while	W and MEI W has not CPU's EAL J's status re e the EALL	DIS only contro been executed LOW/EDIS. Th egister. The MI OW bit in the S	I CLA write acco MEALLOW and is instruction do EALLOW bit in N T1 register only	ess; reads are all d MEDIS are also bes not modify th /ISTF only contro controls access	lowed even if o independant from e EALLOW bit in the ols access for the for the main CPU.				
	As with E of registe	ALLOW, th r accesses	ne MEALLOW I during debug	oit is overridden from Code Com	via the JTAG po poser Studio.	rt, allowing full control				
Flags	This instr	uction does	s not modify fla	gs in the MSTF	register.					
	Flag	TF	ZF	NF	LUF	LVF				
	Modified	NO	No	NO	No	NO				
Pipeline	This is a	single-cycle	e instruction.							
Example	; C heade ; the EPw ;	; C header file including definition of ; the EPwmlRegs structure ;								
	; TNE EPWM TZSEL register is EALLOW protected ;									
	.cdecl	.cdecls C,LIST,"CLAShared.h"								
	_ClalTask	_ClalTask1:								
	MEALLO MMOV16 MEDIS	DW 5 @_EPwmlRe	egs.TZSEL.all,	; Allow CL MR3 ; Write to ; Disallow	A write access TZSEL CLA write acces	ss				
	 MSTOP									
See also	MEDIS									

TEXAS INSTRUMENTS

MEDIS	Disable CLA Write Access to EALLOW Protected Registers								
Operands									
	none	TI	nis instruction doe	s not have any ope	rands				
Opcode	LSW: 0000 0000 0000 MSW: 0111 1111 1011 0000								
Description	This instruction clears the MEALLOW bit in the CLA status register MSTF. When this bit is clear, the CLA is not allowed write access to EALLOW protected registers. To enable CLA writes to protected registers, use the MEALLOW instruction.								
	MEALLOW and MEDIS only control CLA write access; reads are allowed even if MEALLOW has not been executed. MEALLOW and MEDIS are also independant from the main CPU's EALLOW/EDIS. This instruction does not modify the EALLOW bit in the main CPU's status register. The MEALLOW bit in MSTF only controls access for the CLA while the EALLOW bit in the ST1 register only controls access for the main CPU.								
	As with E	ALLOW, th accesses	e MEALLOW during debug	bit is overridder from Code Con	n via the JTAG po nposer Studio.	ort, allowing full control			
Flags	This instruction does not modify flags in the MSTF register.								
	Flag	TF	ZF	NF	LUF				
	woonied	INO	INO	INO	INO	INO			
Pipeline	This is a s	single-cycl	e instruction.						
Example	; C header file including definition of ; the EPwmlRegs structure :								
	; The ePWM TZSEL register is EALLOW protected								
	, .cdecls C,LIST,"CLAShared.h"								
	_ClalTask1:								
	 MEALLOW ; Allow CLA write access MMOV16 @_EPwmlRegs.TZSEL.all, MR3 ; Write to TZSEL MEDIS ; Disallow CLA write access MSTOP								
See also	MEALLO	v							



MEINVF32 MRa, MRb 32-bit Floating-Point Reciprocal Approximation

Operands

	MRa CLA floating-point destination register (MR0 to MR3)							
	MRb	b CLA floating-point source register (MR0 to MR3)						
Opcode	LSW: 0000 0000 bbaa MSW: 0111 1111 0000 0000							
Description	This operation generates an estimate of 1/X in 32-bit floating-point format accurate to approximately 8 bits. This value can be used in a Newton-Raphson algorithm to get a more accurate answer. That is:							
	<pre>Ye = Estimate(1/X); Ye = Ye*(2.0 - Ye*X); Ye = Ye*(2.0 - Ye*X);</pre>							
	After two iterations of the Newton-Raphson algorithm, you will get an exact answer accurate to the 32-bit floating-point format. On each iteration the mantissa bit accuracy approximately doubles. The MEINVF32 operation will not generate a negative zero, DeNorm or NaN value.							
Flags	This instruction modifies the following flags in the MSTF register:							
-	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	No	No	Yes	Yes		
	 The MSTF register flags are modified as follows: LUF = 1 if MEINVF32 generates an underflow condition. LVF = 1 if MEINVF32 generates an overflow condition. 							
Pipeline	This is a s	ingle-cycle	e instruction.					



Example

<pre>; Calculate Num/Den using a ; Ye = Estimate(1/X) ; Ye = Ye*(2.0 - Ye*X) ; Ye = Ye*(2.0 - Ye*X) ;</pre>	a Newton-Raphson algorithum for 1/Der
ClalTask1:	
	; MRl = Den
MEINVF32 MR2, MR1	; MR2 = Ye = Estimate(1/Den)
MMPYF32 MR3, MR2, MR1	; MR3 = Ye*Den
MSUBF32 MR3, #2.0, MR3	; MR3 = 2.0 - Ye*Den
MMPYF32 MR2, MR2, MR3	; MR2 = Ye = Ye*(2.0 - Ye*Den)
MMPYF32 MR3, MR2, MR1	; MR3 = Ye*Den
MMOV32 MR0, @_Num	; MRO = Num
MSUBF32 MR3, #2.0, MR3	; MR3 = 2.0 - Ye*Den
MMPYF32 MR2, MR2, MR3	; MR2 = Ye = Ye*(2.0 - Ye*Den)
MMOV32 MR1, @_Den	; Reload Den To Set Sign
MNEGF32 MR0, MR0, EQ	; if(Den == 0.0) Change Sign Of Num
MMPYF32 MR0, MR2, MR0	; MRO = Y = Ye*Num
MMOV32 @_Dest, MR0	; Store result
MSTOP	; end of task

See also

MEISQRTF32 MRa, MRb



MEISQRTF32 MRa, MRb 32-bit Floating-Point Square-Root Reciprocal Approximation

Operands

	MRa	CLA floating-point destination register (MR0 to MR3)						
	MRb	CLA floating-point source register (MR0 to MR3)						
Opcode	LSW: 0000 MSW: 0111	LSW: 0000 0000 bbaa MSW: 0111 1110 0100 0000						
Description	This operation generates an estimate of 1/sqrt(X) in 32-bit floating-point format accurate to approximately 8 bits. This value can be used in a Newton-Raphson algorithm to get a 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 answer accurate to the 32-bit floating-point format. On each iteration the mantissa bit accuracy approximately doubles. The MEISQRTF32 operation will not generate a negative zero, DeNorm or NaN value.							
Flags	This instru	iction modi	fies the followi	ng flags in the M	ISTE register			
i lago	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	No	No	Yes	Yes		
	 The MSTF register flags are modified as follows: LUF = 1 if MEISQRTF32 generates an underflow condition. LVF = 1 if MEISQRTF32 generates an overflow condition. 							
Pipeline	This is a s	single-cycle	instruction.					


Example

<pre>; Y = sqrt(X) ; Ye = Estimate(1/sqrt(X)); ; Ye = Ye*(1.5 - Ye*Ye*X*0.5) ; Ye = Ye*(1.5 - Ye*Ye*X*0.5) ; Y = X*Ye ;</pre>		
_Cla1Task3:		
MMOV32 MR0, @_x	;	MR0 = X
MEISQRTF32 MR1, MR0	;	<pre>MR1 = Ye = Estimate(1/sqrt(X))</pre>
MMOV32 MR1, @_x, EQ	;	if(X == 0.0) Ye = 0.0
MMPYF32 MR3, MR0, #0.5	;	MR3 = X*0.5
MMPYF32 MR2, MR1, MR3	;	MR2 = Ye * X * 0.5
MMPYF32 MR2, MR1, MR2	;	MR2 = Ye*Ye*X*0.5
MSUBF32 MR2, #1.5, MR2	;	MR2 = 1.5 - Ye*Ye*X*0.5
MMPYF32 MR1, MR1, MR2	;	$MR1 = Ye = Ye^{(1.5 - Ye^{Xe^{X}0.5)}$
MMPYF32 MR2, MR1, MR3	;	MR2 = Ye * X * 0.5
MMPYF32 MR2, MR1, MR2	;	MR2 = Ye*Ye*X*0.5
MSUBF32 MR2, #1.5, MR2	;	MR2 = 1.5 - Ye*Ye*X*0.5
MMPYF32 MR1, MR1, MR2	;	$MR1 = Ye = Ye^{(1.5 - Ye^{X*0.5})}$
MMPYF32 MR0, MR1, MR0	;	MR0 = Y = Ye*X
MMOV32 @_y, MR0	;	Store $Y = sqrt(X)$
MSTOP	;	end of task

See also

MEINVF32 MRa, MRb



MF32TOI16 MRa, MRb Convert 32-bit Floating-Point Value to 16-bit Integer

	MRa	CLA flo	ating-point destina	tion register (M	R0 to MR3)				
	MRb CLA floating-point source register (MR0 to MR3)								
Opcode	LSW: 0000 MSW: 0111	0000 0000 bba 1101 1110 000	aa)0						
Description	Convert a 3 will be store	32-bit floating ed in MRa.	point value in	MRb to a 16	-bit integer and t	runcate. The resu	lt		
	MRa(15:0) MRa(31:16)	= F32TOI16(MF = sign exter	Rb); nsion of MRa(1	5);					
Flags	This instruc	tion does not	affect any flag	IS:					
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No			
Pipeline	This is a si	ngle-cycle ins	truction.						
Example	MMOVIZ MF32TOI16	MMOVIZ MR0, #5.0 ; MR0 = 5.0 (0x40A00000) MF32TOI16 MR1, MR0 ; MR1(15:0) = MF32TOI16(MR0) = 0x0005 ; MR1(31:16) = Sign extension of MR1(15) = 0x0000							
	MMOVIZ MF32TOI16	MMOVIZ MR2, #-5.0 ; MR2 = -5.0 (0xC0A00000) MF32TOI16 MR3, MR2 ; MR3(15:0) = MF32TOI16(MR2) = -5 (0xFFFB) ; MR3(31:16) = Sign extension of MR3(15) = 0xFFFF							
See also	MF32TOI16R MRa, MRb MF32TOUI16 MRa, MRb MF32TOUI16R MRa, MRb MI16TOF32 MRa, MRb MI16TOF32 MRa, mem16 MUI16TOF32 MRa, mem16								
	MUI16TOF	MUI16TOF32 MRa, MRb							



MF32TOI16R MRa, MRb Convert 32-bit Floating-Point Value to 16-bit Integer and Round

nd round to the near	rest						
nd round to the near	rest						
nd round to the near	rest						
nd round to the near	rest						
LVF							
No							
MMOVIZ MR0, $\#0x3FD9$; MR0(31:16) = 0x3FD9 MMOVXI MR0, $\#0x999A$; MR0(15:0) = 0x999A ; MR0 = 1.7 (0x3FD9999A)							
MF32TOI16R MR1, MR0 ; MR1(15:0) = MF32TOI16round (MR0) = 2 (0x0002)							
$) = 0 \times 0 0 0 0$							
MMOVF32 MR2, #-1.7 ; MR2 = -1.7 (UXBFD9999A) MF32TOI16R MR3, MR2 ; MR3(15:0) = MF32TOI16round (MR2) = -2 (0xFFFE)							
$) = 0 \times FFFF$							
	LVF No (0x0002)) = 0x0000 : (0xFFFE)) = 0xFFFF						



MF32TOI32 MRa, MRb Convert 32-bit Floating-Point Value to 32-bit Integer

	MRa	MRa CLA floating-point destination register (MR0 to MR3)								
	MRb	CL	A floating-point sc	urce register (MR0	to MR3)					
Opcode	LSW: 0000 MSW: 0111	0000 0000 1101 0110	bbaa 0000							
Description	Convert the Store the re MRa = F32T	e 32-bit flo esult in M	oating-point val Ra. ;	ue in MRb to a	32-bit integer val	ue and truncate.				
Flags	This instruc	This instruction does not affect any flags:								
	Flag Modified	TF No	ZF	NF	LUF	LVF No				
Pipeline	This is a si	ngle-cycle	e instruction.							
Example 1	MMOVF32 MF32T0I32 MMOVF32 MF32T0I32	MR2, #11 MR3, MR2 MR0, #-1 MR1, MR0	204005.0 ; N ; N 1204005.0 ; N ; N	IR2 = 11204005. IR3 = MF32TOI32 IR0 = -11204005 IR1 = MF32TOI32	0 (0x4B2AF5A5) (MR2) = 1120400 .0 (0xCB2AF5A5) (MR0) = -1120400	5 (0x00AAF5A5) 05 (0xFF550A5B)				
Example 2	<pre>; Given X, ; X = IQ24 ; M = IQ24 ; B = IQ24 ; ; Calculat ; ; Convert ; ; ClalTask2 MI32TOF MI32TOF MI32TOF MI32TOF MMPYF32 MMPYF32 MADDF32 ; Convert MMPYF32 MADDF32</pre>	<pre>M and B (+2.5) = (+1.5) = (-0.5) = e Y = X * M, X and : 32 MR0, @ 32 MR1, @ 32 MR2, @ MR0, M MR1, M MR2, M MR2, M MR2, M Y from fl MR2, MR2 32 MR2, M @_Y, MR2</pre>	are IQ24 numbe 0x02800000 0x01800000 0xFF800000 M + B B from IQ24 to 	<pre>mrs: mr0 = 0x4BC000 Mr1 = 0x4C2000 Mr2 = 0xCB0000 M = 1/(1*2^24) X = 1/(1*2^24) X = 1/(1*2^24) M*X Y=MX+B = 3.25 Y * 1*2^24 IQ24(Y) = 0x03 store result end of task</pre>	00 00 00 * iqm = 1.5 (0: * iqx = 2.5 (0: * iqb =5 (0: (0x40500000) 400000	<3FC00000) <40200000) <bf000000)< td=""></bf000000)<>				
See also	MF32TOUI MI32TOF3 MI32TOF3 MUI32TOF MUI32TOF	32 MRa, 2 MRa, M 2 MRa, m 32 MRa, 32 MRa,	MRb Rb em32 MRb mem32							



MF32TOUI16 MRa, MRb Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer

	MRa	CLA flo	ating-point de	stination register (M	R0 to MR3)		
	MRb CLA floating-point source register (MR0 to MR3)						
Opcode	LSW: 0000 0 MSW: 0111 1	0000 0000 bba .110 1010 000	a 0				
Description	Convert the truncate to a nearest eve MRa(15:0) = MRa(31:16)	a 32-bit floatin zero. The res on value use t = F32TOUI16(M = 0x0000;	g point valu ult will be s he MF32T(^{Rb);}	ue in MRb to an tored in MRa. T DUI16R instruct	unsigned 16-bit o instead round ion.	integer value and the integer to the	
Flags	This instruc	tion does not	affect any	flags:			
	Flag	TF	ZF	NF	LUF		
	Modified	INU	INU	INU	INO	INO	
Pipeline	This is a sir	ngle-cycle ins	truction.				
Example	MMOVIZ MF32TOUI16 MMOVIZ MF32TOUI16	MR0, #9.0 MR1, MR0 MR2, #-9.0 MR3, MR2	; MR0 = ; MR1(15 ; MR1(31 ; MR2 = ; MR3(15 ; MR3(31	9.0 (0x41100000 :0) = MF32TOUI :16) = 0x0000 -9.0 (0xC110000 :0) = MF32TOUI :16) = 0x0000)) 16(MR0) = 9 (0x) 00) 16(MR2) = 0 (0x)	0009)	
See also	MF32TOI16 MF32TOUI MF32TOUI MI16TOF32 MI16TOF32 MUI16TOF3 MUI16TOF3	MRa, MRb 16R MRa, MF 16R MRa, MF 2 MRa, MRb 2 MRa, mem 32 MRa, men 32 MRa, MRb	Rb Rb 6 116				

MF32TOUI16R MRa, MRb Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer and Round

	MRa	CLA floating	g-point destination register (M	IR0 to MR3)					
	MRb	CLA floating	g-point source register (MR0	to MR3)					
Opcode	LSW: 0000 0 MSW: 0111 1	000 0000 bbaa 110 1100 0000							
Description	Convert the the closest e converted va	Convert the 32-bit floating-point value in MRb to an unsigned 16-bit integer and round to the closest even value. The result will be stored in MRa. To instead truncate the converted value, use the MF32TOUI16 instruction.							
	MRa(15:0) MRa(31:16)	MRa(15:0) = MF32TOUI16round(MRb); MRa(31:16) = 0x0000;							
Flags	This instruct	ion does not aff	ect any flags:						
	Flag	TF ZF	F NF	LUF	LVF				
	Modified	No No	o No	No	No				
Pipeline	This is a sin	gle-cycle instruc	tion.						
Pipeline Example	MMOVIZ MMOVXI MF32TOUI16R	MR0, #0x412C MR0, #0xCCCD MR1, MR0	; MR0 = 0x412C ; MR0 = 0xCCCD ; MI ; MR1(15:0) = MF32' ; MR1(31:16) = 0xO ; MR2 = 10.0 (0-20)	R0 = 10.8 (0x412 TOUI16round(MR0)	2CCCCD)) = 11 (0x000B)				
	MMOVF32 MF32TOUI16R	MR2, #-10.8 MR3, MR2	; MR2 = -10.8 (0X0) ; MR3(15:0) = MF32 ; MR3(31:16) = 0x0	COUILGround(MR2)	$) = 0 (0 \times 0000)$				
See also	MF32TOI16 MF32TOI16 MF32TOUI1 MI16TOF32 MI16TOF32 MUI16TOF3 MUI16TOF3	MRa, MRb R MRa, MRb 6 MRa, MRb MRa, MRb MRa, mem16 2 MRa, mem16 2 MRa, MRb							



MF32TOUI32 MRa, MRb Convert 32-bit Floating-Point Value to 16-bit Unsigned Integer

•	MRa	CLA flo	pating-point dest	ination register (M	R0 to MR3)	
	MRb	CLA flo	pating-point sour	ce register (MR0 t	to MR3)	
Opcode	LSW: 0000 (MSW: 0111 1	0000 0000 bb 1101 1010 00	aa 00			
Description	Convert the result in MF	e 32-bit floatii Ra.	ng-point value	e in MRb to an	unsigned 32-bit	integer and store the
	MRa = F32TC	OUI32(MRb);				
Flags	This instruc	tion does no	t affect any fl	ags:		
	Flag	TF	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No
Pipeline	This is a sir	ngle-cycle ins	struction.			
Example	MMOVIZ MF32TOUI32 MMOVIZ MF32TOUI32	MR0, #12.5 MR0, MR0 MR1, #-6.5 MR2, MR1	; MR0 = 1 ; MR0 = M ; MR1 = - ; MR2 = M	2.5 (0x414800) F32TOUI32 (MR(6.5 (0xC0D000) F32TOUI32 (MR)	00) 0) = 12 (0x00000 00) 1) = 0.0 (0x0000	000C) 00000)
See also	MF32TOI32 MI32TOF32 MI32TOF32 MUI32TOF3 MUI32TOF3 MUI32TOF3	2 MRa, MRb 2 MRa, MRb 2 MRa, mem 32 MRa, MR 32 MRa, mei	32 b m32			



MFRACF32 MRa, MRb Fractional Portion of a 32-bit Floating-Point Value

eperanae							
	MRa	CL	A floating-point de	stination register (M	R0 to MR3)		
	MRb	CL	A floating-point so	urce register (MR0 t	o MR3)		
Opcode	LSW: 0000 MSW: 0111	0000 0000 1110 0000	bbaa 0000				
Description	Returns ir	MRa the f	ractional portio	n of the 32-bit fl	oating-point valu	ie in MRb	
Flags	This instru	uction does	not affect any	flags:			
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a s	single-cycle	instruction.				
Example	MMOVIZ MFRACF32	MR2, #19.0 MR3, MR2	525 ; MR2 = 19 ; MR3 = MF	.625 (0x419D000 RACF32(MR2) = ()0)).625 (0x3F2000()0)0)	
See also							



MI16TOF32 MRa, MRb Convert 16-bit Integer to 32-bit Floating-Point Value

	MRa	CLA floa	ting-point de	stination register (MR	0 to MR3)		
	MRb	CLA floa	iting-point so	urce register (MR0 to	MR3)		
Opcode	LSW: 0000 MSW: 0111	0000 0000 bbaa 1110 1000 000	a 0				
Description	Convert the result in MF	e 16-bit signed Ra.	integer in	MRb to a 32-bit	floating point va	alue and store the	е
Flags	This instruc	ction does not	affect any	flags:			
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a si	ngle-cycle inst	ruction.				
Example	MMOVIZ MMOVXI MI16TOF32	MR0, #0x000 MR0, #0x000 MR1, MR0) ; MR0 4 ; MR0 ; MR1	0(31:16) = 0.0 (0 0(15:0) = 4.0 (0 . = MI16TOF32 (MF	(0x0000) (x0004) (x0) = 4.0 (0x40))800000)	
	MMOVIZ MMOVXI MI16TOF32 MSTOP	MR2, #0x000 MR2, #0xFFF MR3, MR2) ; MR2 C ; MR2 ; MR3	2(31:16) = 0.0 (0 2(15:0) = -4.0 (0 3 = MI16TOF32 (MF	0x0000) 0xFFFC) R2) = -4.0 (0x0	20800000)	
See also	MF32TOI10 MF32TOI10 MF32TOUI MF32TOUI MI16TOF33 MUI16TOF MUI16TOF	6 MRa, MRb 6R MRa, MRb 16 MRa, MRb 16R MRa, MR 2 MRa, mem1 32 MRa, mem 32 MRa, MRb	b 6 16				



MI16TOF32 MRa, mem16 Convert 16-bit Integer to 32-bit Floating-Point Value

•	MRa	CL	A floating-point de	stination register (N	IR0 to MR3)		
	mem16	16	-bit source memor	y location to be con	verted		
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm 0101 00aa	mmmm addr				
Description	Convert th floating-po MRa = MI1	ne 16-bit si pint value a 6TOF32[mem	gned integer in and store the re	dicated by the n esult in MRa.	nem16 pointer to	a 32-bit	
Flags	This instru	ction does	not affect any	flags:			
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a s	ingle-cycle	e instruction:				
Example	; Assume ; ;	A = 4 (0x0 B = -4 (0x F32 MR0, @	004) FFFC) _A ; MR0 = MII	L6TOF32(A) = 4.	0 (0x40800000)		
	MI16TO	F32 MR1, @	_B ; MR1 = MI1	16TOF32(B) = -4	.0 (0xC0800000		
See also	MF32TOI MF32TOL MF32TOL MF32TOL MI16TOF MUI16TO MUI16TO	16 MRa, M 16R MRa, JI16 MRa, JI16R MRa 32 MRa, M F32 MRa, F32 MRa,	Rb MRb MRb , MRb Rb mem16 MRb				



MI32TOF32 MRa, mem32 Convert 32-bit Integer to 32-bit Floating-Point Value

operando	MRa CLA floating-point destination register (MR0 to MR3)								
	mem32	32-	bit memory s	ource for the MMOV32	operation.				
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm 0100 01aa	mmmm addr						
Description	Convert the store the r	ne 32-bit sig result in MR	ned intege a.	er indicated by mem	132 to a 32-bit flo	ating point value) and		
	<pre>MRa = MI32TOF32[mem32];</pre>								
Flags	This instruction does not affect any flags:								
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No			
Pipeline	This is a s	single-cycle	instructior	1.					
Example	; Given X ; X = IQ2 ; M = IQ2 ; B = IQ2 ; ; Calcula ; ; Convert ;	<pre>, M and B a 4(+2.5) = C 4(+1.5) = C 4(-0.5) = C te Y = X * M, X and E</pre>	re IQ24 n x02800000 x01800000 xFF800000 M + B from IQ2	umbers: 4 to float					
	ClalTask MI32TO MI32TO MI32TO MMPYF3 MMPYF3 MMPYF3 MADDF3 ; Convert MMPYF3 MF32TO MMOV32 MSTOP	3: F32 MR0, @ F32 MR1, @_ F32 MR2, @_ 2 MR0, MR0, 2 MR1, MR1, 2 MR2, MR2, 2 MR2, MR2, Y from flc 2 MR2, MR2, I32 MR2, MR2	M x B #0x3380 #0x3380 MR1 MR3 vat32 to I #0x4B80 .2	<pre>; MR0 = 0x4BC0000 ; MR1 = 0x4C20000 ; MR2 = 0xCB00000 ; M = 1/(1*2^24) ; X = 1/(1*2^24) ; B = 1/(1*2^24) ; M*X ; Y=MX+B = 3.25 (024 ; Y * 1*2^24 ; IQ24(Y) = 0x034 ; store result ; end of task</pre>	0 0 * iqm = 1.5 (0x: * iqx = 2.5 (0x: * iqb =5 (0x) 0x40500000)	3FC00000) 40200000) 3F000000)			
See also	MF32TOI MF32TOU MI32TOF MUI32TO MUI32TO MUI32TO	32 MRa, MF JI32 MRa, M 32 MRa, MF F32 MRa, M F32 MRa, n	Rb IRb Rb IRb nem32						



MI32TOF32 MRa, MRb Convert 32-bit Integer to 32-bit Floating-Point Value

	MRa CLA floating-point destination register (MR0 to MR3)						
	MRb	CL	A floating-point so	urce register (MR0 t	o MR3)		
Opcode	LSW: 0000 MSW: 0111	0000 0000 1101 1000	bbaa 0000				
Description	Convert th result in N MRa = MI3	ne signed 3 1Ra. 2TOF32(MRb	32-bit integer in	MRb to a 32-bit	floating-point va	alue and store the	
Flags	This instru	uction does	not affect any	flags:			
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a s	single-cycle	e instruction.				
Example	; Example: ; MMOVIZ MMOVXI MI32TO	1: MR2, # MR2, # F32 MR3, M	0x1111 ; MR2(3 0x1111 ; MR2(1 ; MR2 = R2 ; MR3 =	31:16) = 4369 (0 15:0) = 4369 (0 = +286331153 (0 = MI32TOF32 (MR2	0x1111) x1111) x11111111) 2) = 286331153.0) (0x4D888888)	
See also	MF32TOI MF32TOL MI32TOF MUI32TO MUI32TO MUI32TO	32 MRa, M JI32 MRa, 32 MRa, m F32 MRa, F32 MRa, F32 MRa,	Rb MRb em32 MRb mem32				

MLSL32 MRa, #SHIFT Logical Shift Left

•	MRa	CLA	floating-point sou	rce/destination regi	ster (MR0 to MR3)		
	#SHIFT	Num	ber of bits to shift	t (1 to 32)			
Opcode	LSW: 0000 MSW: 0111	0000 0shi f 1011 1100 (Etaa 0000				
Description	Logical sh 32. MARa(31:0	ift left of MR	Shift Left(M2	Der of bits indica	ated. The numbe	er of bits can be 1 to	
Flags	This instru	ction modifi	es the followin	g flags in the M	STF register:		
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	Yes	Yes	No	No	
	Ine MSIF NF = MRa(1 ZF = 0; if(MRa(31	<pre>- register flag 31); :0) == 0) {</pre>	<pre>gs are modifie zF = 1; }</pre>	d based on the	integer results c	of the operation.	
Pipeline	This is a s	ingle-cycle i	nstruction.				
Example	<pre>initial is a single-cycle instruction. ; Given m2 = (int32)32 ; x2 = (int32)64 ; b2 = (int32)-128 ; ; Calculate: ; m2 = m2*2 ; x2 = x2*4 ; b2 = b2*8 ;</pre>						
See also	MSTOP ; end of task MADD32 MRa, MRb, MRc MASR32 MRa, #SHIFT MAND32 MRa, MRb, MRc MLSR32 MRa, MRb, MRc MXOR32 MRa, MRb, MRc MXOR32 MRa, MRb, MRc MSUB32 MRa, MRb, MRc						

MLSR32 MRa, #SHIFT Logical Shift Right

-	MRa	a CLA floating-point source/destination register (MR0 to MR3)								
	#SHIFT	Nu	mber of bits to shi	t (1 to 32)						
Opcode	LSW: 0000 00 MSW: 0111 10	00 0shi 11 1000	ftaa 0000							
Description	Logical shift 32. Unlike th sign bit. Even vacant bit-po MARa(31:0) =	right of l e arithm ry bit in sitions a Logica	MRa by the nur letic shift (MAS the operand is are filled in with 1 Shift Right(nber of bits india R32), the logica moved the spec zeros MARa(31:0) by #	cated. The numb I shift does not p ified number of k #SHIFT bits);	er of bits can be 1 to reserve the number's bit positions, and the				
Flags	This instructi	on modi	fies the followir	ng flags in the M	ISTF register:					
	Flag	TF	ZF	NF	LUF	LVF				
	Modified	No	Yes	Yes	No	No				
Pipeline	This is a sing	This is a single-cycle instruction.								
Pipeline	This is a sing	gle-cycle	instruction.							
Example	; Illustrate the difference between MASR32 and MLSR32									
	MMOVIZ MR0, MMOVXI MR0,	#0xAAAA #0x5555	; MRO = 0xA	AAA5555						
	MMOV32 MR1,	MR0	; MR1 = 0xA	AAA5555						
	MMOV32 MR2,	MR0	; MR2 = $0xA$	AAA5555						
	MASR32 MR1,	#1	; MR1 = $0xD$	5552AAA						
	MLSR32 MR2,	#1	; MR2 = $0x5$	5552AAA						
	MASR32 MR1,	#1	; MR1 = 0xE	AAA9555						
	MLSR32 MR2,	#1	; MR2 = $0x2$	AAA9555						
	MASR32 MR1,	#6	; MR1 = $0 \times F$	FAAAA55						
	MLSR32 MR2,	#b	; MRZ = 0X0	UAAAA55						
See also	MADD32 MF	Ra, MRb	, MRc							
		a, #SHI 2a MRh								
	MLSL32 MR	a. #SHI	T							
	MOR32 MRa	a, MRb,	MRc							
	MXOR32 MF	Ra, MRb	, MRc							
	MSUB32 MR	la, MRb	, MRc							

MMACF32 MR3, MR2, MRd, MRe, MRf ||MMOV32 MRa, mem32 32-bit Floating-Point Multiply and Accumulate with Parallel Move

Operands									
	MR3	floa	ating-point destinat	ion/source register N	AR3 for the add ope	ration			
	MR2	CLA floating-point source register MR2 for the add operation							
	MRd	CL. MR	A floating-point dead dead the state of the	stination register (MI ame register as MRa	R0 to MR3) for the m a	nultiply operation			
MRe CLA floating-point source register (MR0 to MR3) for the multiply operation									
	MRf CLA floating-point source register (MR0 to MR3) for the multiply operation								
MRa CLA floating-point destination register for the MMOV32 operation MRa cannot be MR3 or the same register as MRd.						tion (MR0 to MR3).			
	mem32	32-	bit source for the I	MMOV32 operation					
Opcode	LSW: mmmm m MSW: 0011 f	mmm mmmm fee ddaa	mmmm addr						
Description	Multiply and to memory. destination MR3 = MR3 + MRd = MRe * MRa = [mem3	Multiply and accumulate the contents of floating-point registers and move from register to memory. The destination register for the MMOV32 cannot be the same as the destination registers for the MMACF32. MR3 = MR3 + MR2; MRd = MRe * MRf; MRa = [mem32];							
Restrictions	The destina MRa canno	ition regis t be MR3	ters for the MM and MRa canr	IACF32 and the not be the same	MMOV32 must register as MRd	be unique. That is,			
Flags	This instruc	tion modi	fies the followir	ng flags in the M	STF register:				
C	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	Yes	Yes	Yes	Yes			
	The MSTF (• LUF = 1 • LVF = 1 MMOV32 so NF = MRa(31 ZF = 0;	register fla if MMAC if MMAC ets the NF	ags are modifie F32 (add or mi F32 (add or mi F and ZF flags	ed as follows: ultiply) generates ultiply) generates as follows:	s an underflow c s an overflow co	ondition. ndition.			
Dinalina	if(MRa(30:2	(3) == 0)	$\{ ZF = 1; NF \}$	= 0; }					
Pipeline	MMACE32 and MMOV32 complete in a single cycle.								



Instruction	Set
111311 4611011	000

Example 1

;	Perform 5 multiply and accumulate	e operations:
;	X and Y are 32-bit floating poin	t arrays
, ; ; ; ; ; .	<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 = $A + B + C + D + E$	
; _C	<pre>lalTask1: MMOVI16 MAR0, #_X MMOVI16 MAR1, #_Y MNOP MMOV32 MR0, *MAR0[2]++ MMOV32 MR1, *MAR1[2]++</pre>	<pre>; MAR0 points to X array ; MAR1 points to Y array ; Delay for MAR0, MAR1 load ; Delay for MAR0, MAR1 load ; < MAR0 valid ; MR0 = X0, MAR0 += 2 ; < MAR1 valid ; MR1 = Y0, MAR1 += 2</pre>
	MMPYF32 MR2, MR0, MR1 MMOV32 MR0, *MAR0[2]++ MMOV32 MR1, *MAR1[2]++	; MR2 = A = X0 * Y0 ; In parallel MR0 = X1, MAR0 += 2 ; MR1 = Y1, MAR1 += 2
	MMPYF32 MR3, MR0, MR1 MMOV32 MR0, *MAR0[2]++ MMOV32 MR1, *MAR1[2]++	<pre>; MR3 = B = X1 * Y1 ; In parallel MR0 = X2, MAR0 += 2 ; MR1 = Y2, MAR2 += 2</pre>
	MMACF32 MR3, MR2, MR2, MR0, MR1 MMOV32 MR0, *MAR0[2]++ MMOV32 MR1, *MAR1[2]++	; MR3 = A + B, MR2 = C = X2 * Y2 ; In parallel MR0 = X3 ; MR1 = Y3 M
	MACF32 MR3, MR2, MR2, MR0, MR1 MMOV32 MR0, *MAR0 MMOV32 MR1, *MAR1	<pre>; MR3 = (A + B) + C, MR2 = D = X3 * Y3 ; In parallel MR0 = X4 ; MR1 = Y4</pre>
	MMPYF32 MR2, MR0, MR1 MADDF32 MR3, MR3, MR2	; MR2 = E = X4 * Y4 ; in parallel MR3 = (A + B + C) + D
	MADDF32 MR3, MR3, MR2 MMOV32 @_Result, MR3 MSTOP	; MR3 = (A + B + C + D) + E ; Store the result ; end of task



Example 2	<pre>; sum = X0*B0 + X1*B1 + X2*B2 + Y1*A1 + Y2*B2 ; ; X2 = X1 ; X1 = X0 ; Y2 = Y1 ; Y1 = sum</pre>							
	; _ClaTask2: MMOV32 MR0, @_B2 ; MR0 = B2 MMOV32 MR1, @_X2 ; MR1 = X2 MMPYF32 MR2, MR1, MR0 ; MR2 = X2*B2 MMOV32 MR0, @_B1 ; MR0 = B1 MMOVD32 MR1, @_X1 ; MR1 = X1, X2 = X1 MMPYF32 MR3, MR1, MR0 ; MR3 = X1*B1 MMOV32 MR0, @_B0 ; MR0 = B0 MMOVD32 MR1, @_X0 ; MR1 = X0, X1 = X0							
	<pre>; MR3 = X1*B1 + X2*B2, MR2 = X0*B0 ; MR0 = A2 MMACF32 MR3, MR2, MR2, MR1, MR0 MMOV32 MR0, @_A2 M MOV32 MR1, @_Y2 ; MR1 = Y2</pre>							
	<pre>; MR3 = X0*B0 + X1*B1 + X2*B2, MR2 = Y2*A2 ; MR0 = A1 MMACF32 MR3, MR2, MR2, MR1, MR0 MMOV32 MR0, @_A1</pre>							
	MMOVD32 MR1,@_Y1; MR1 = Y1, Y2 = Y1MADDF32 MR3, MR3, MR2; MR3 = Y2*A2 + X0*B0 + X1*B1 + X2*B2 MMPYF32 MR2, MR1, MR0; MR2 = Y1*A1MADDF32 MR3, MR3, MR2; MR3 = Y1*A1 + Y2*A2 + X0*B0 + X1*B1 + X2*B2MMOV32 @_Y1, MR3; Y1 = MR3MSTOP; end of task							

See also

MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MMAXF32 MRa, MRb 32-bit Floating-Point Maximum

	MRa	CLA f	oating-point s	source/destination regis	ter (MR0 to MR3)					
	MRb	CLA f	oating-point s	source register (MR0 to	MR3)					
Opcode	LSW: 0000 C MSW: 0111 1	0000 0000 bk 1101 0010 00)00							
Description	if(MRa < MF	Rb) MRa = MF	lb;							
	Special cas NaN out A denor 	es for the output will be of malized outp	utput from converted to out will be o	the MMAXF32 ope o infinity converted to positi	eration: ve zero.					
Flags	This instruc	This instruction modifies the following flags in the MSTF register:								
	Flag	TF	ZF	NF	LUF	LVF				
	Modified	No	Yes	Yes	No	No				
	The ZF and in the destin if(MRa == M if(MRa > MF if(MRa < MF	INF flags an nation regist MRb) {ZF=1; Rb) {ZF=0; M Rb) {ZF=0; M	<pre>re configure er. NF=0;} IF=0;} IF=1;}</pre>	ed on the result of	the operation, r	not the result sto	ored			
Pipeline	This is a sir	ngle-cycle in	struction.							
Example 1	$\begin{array}{llllllllllllllllllllllllllllllllllll$									
Example 2	; X is an a ; Find the ; and store ; _ClalTask1: MMOV16 MU16TOF3 MNOP MMOV32 LOOP MMOV32 LOOP MMOV32 MMAXF32 MADDF32 MCMPF32 MNOP MNOP MNOP MNOP MNOP MNOP MNOP MNOP	Array of 32- maximum val e it in Resu MAR1,#_X 32 MR0, @_le MR1, *MAH MR2, *MAH MR1, MR2 MR0, MR0, MR0, MR0, MR0 #0.0 LOOP, NE(@_Result,	bit floati ue in an a alt en en el[2]++ #-1.0	ing-point values array X Start address Length of the ar delay for MAR1 : delay for MAR1 : delay for MAR1 : MR1 = X0 MR2 = next eleme MR1 = MAX(MR1, M Decrement the co Set/clear flags Branch if not ed Always executed Always executed Always executed End of task	rray load load ent MR2) bunter for MBCNDD qual to zero					
See also	MCMPF32 MCMPF32 MMAXF32 MMINF32 M MMINF32 M	MRa, MRb MRa, #16FI MRa, #16FI MRa, MRb /Ra, #16FH	Hi Hi							



MMAXF32 MRa, #16FHi 32-bit Floating-Point Maximum

•p•••••	MRa	CLA	floating-point sour	ce/destination reg	ister (MR0 to MR3)			
	#16FHi	A 16 floati	-bit immediate valung-point value. Th	ue that represents e low 16-bits of th	the upper 16-bits of a e mantissa are assum	n IEEE 32-bit ned to be all 0.		
Opcode	LSW: IIII I MSW: 0111 1	III IIII I LOO1 0000 C	III Oaa					
Description	Compare M immediate	Ra with the	e floating-point ger, then load i	value represe t into MRa.	nted by the imme	diate operand. If the		
#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IE floating-point value. The low 16-bits of the mantissa are assumed to be all 0. addressing mode is most useful for constants where the lowest 16-bits of the are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F00 -1.5 (0xBFC00000). The assembler will accept either a hex or float as the im value. That is, -1.5 can be represented as #-1.5 or #0xBFC0						of an IEEE 32-bit be all 0. This its of the mantissa (0x3F000000), and is the immediate		
	 Special cases for the output from the MMAXF32 operation: NaN output will be converted to infinity A denormalized output will be converted to positive zero. 							
Flags	This instruc	tion modifie	es the following	g flags in the M	ISTF register:			
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	Yes	Yes	No	No		
	<pre>The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register. if(MRa == #16FHi:0) {ZF=1; NF=0;} if(MRa > #16FHi:0) {ZF=0; NF=0;} if(MRa < #16FHi:0) {ZF=0; NF=1;}</pre>							
Pipeline	This is a sir	ngle-cycle i	nstruction.					
Example	MMOVIZ MR0, #5.0 ; MR0 = 5.0 (0x40A00000) MMOVIZ MR1, #4.0 ; MR1 = 4.0 (0x40800000) MMOVIZ MR2, #-1.5 ; MR2 = -1.5 (0xBFC00000) MMAXF32 MR0, #5.5 ; MR0 = 5.5, ZF = 0, NF = 1 MMAXF32 MR1, #2.5 ; MR1 = 4.0, ZF = 0, NF = 0 MMAXF32 MR2, #-1.0 ; MR2 = -1.0, ZF = 0, NF = 1 MMAXF32 MR2, #-1.0 ; MR2 = -1.5, ZF = 1, NF = 0							
See also	MMAXF32 MMINF32 M MMINF32 M	MRa, MRb /IRa, MRb /IRa, #16Ft	Hi					

MMINF32 MRa, MRb 32-bit Floating-Point Minimum

Operando										
	MRa	CLA floating-	point source/destination	n register (MR0 to MR3)						
	MRb	CLA floating-	point source register (N	/IR0 to MR3)						
Opcode	LSW: 0000 0000 MSW: 0111 1101	0000 bbaa 0100 0000								
Description	if(MRa > MRb)	MRa = MRb;								
	Special cases	for the output	from the MMINF32	operation:						
	NaN output	will be conver	ted to infinity							
	A denormal	iized output wi	Il be converted to p	positive zero.						
Flags	This instruction	This instruction modifies the following flags in the MSTF register:								
	Modified No	D Yes	s Yes	No	No					
	The ZF and NF in the destinati if(MRa == MRb) if(MRa > MRb) if(MRa < MRb)	flags are con on register. {zF=1; NF=0 {zF=0; NF=0;} {zF=0; NF=1;}	figured on the resu	ult of the operation, i	not the result stored					
Pipeline	This is a single	-cycle instruct	ion.							
Example 1	MMOVIZ MR0, #5.0 ; MR0 = 5.0 (0x40A00000) MMOVIZ MR1, #4.0 ; MR1 = 4.0 (0x4080000) MMOVIZ MR2, #-1.5 ; MR2 = -1.5 (0xBFC00000) MMINF32 MR0, MR1 ; MR0 = 4.0, ZF = 0, NF = 0 MMINF32 MR1, MR2 ; MR1 = -1.5, ZF = 0, NF = 0 MMINF32 MR2, MR1 ; MR2 = -1.5, ZF = 1, NF = 0 MMINF32 MR1, MR0 ; MR2 = -1.5, ZF = 0, NF = 1									
Example 2	<pre>MMINF32 MR1, MR0 ; MR2 = -1.5, ZF = 0, NF = 1 ; ; X is an array of 32-bit floating-point values ; Find the minimum value in an array X ; and store it in Result ; </pre>									
See also	MMAXF32 MR MMAXF32 MR MMINF32 MRa	a, MRb a, #16FHi a, #16FHi								



MMINF32 MRa, #16FHi 32-bit Floating-Point Minimum

operando	MRa	floating	a-point source/des	tination register (I	MR0 to MR3)					
	#16FHi	A 16-b floating	it immediate value g-point value. The	ediate value that represents the upper 16-bits of an IEEE 32-bit value. The low 16-bits of the mantissa are assumed to be all 0.						
Opcode	LSW: IIII IIII IIII IIII MSW: 0111 1001 0100 00aa									
Description	Compare MR immidate val	a with the ue is smalle	floating-point ver, then load it	value represer into MRa.	nted by the imme	ediate operand. If the				
	if(MRa > #16	FHi:0) MRa	= #16FHi:0;							
	#16FHi is a 1 floating-point addressing m are 0. Some -1.5 (0xBFC0 value. That is	#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 $_{1.5}$ can be represented as $_{1.5}$ cr $_{1.5}$ can be represented as $_$								
	Special cases for the output from the MMINE32 operation:									
	 NaN output will be converted to infinity A denormalized output will be converted to positive zero. 									
Flags	This instruction	on modifies	the following	flags in the M	STF register:					
-	Flag	TF	ZF	NF	LUF	LVF				
	Modified	No	Yes	Yes	No	No				
	The ZF and I in the destina if(MRa == #1 if(MRa > #16 if(MRa < #16	The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register. if(MRa == #16FHi:0) {ZF=1; NF=0;} if(MRa > #16FHi:0) {ZF=0; NF=0;} if(MRa < #16FHi:0) {ZF=0; NF=1;}								
Pipeline	This is a sing	le-cycle ins	struction.							
Example	MMOVIZ MR0 MMOVIZ MR1 MMOVIZ MR2 MMINF32 MR0 MMINF32 MR1 MMINF32 MR2 MMINF32 MR2	MMOVIZ MR0, #5.0 ; MR0 = 5.0 (0x40A00000) MMOVIZ MR1, #4.0 ; MR1 = 4.0 (0x40800000) MMOVIZ MR2, #-1.5 ; MR2 = -1.5 (0xBFC00000) MMINF32 MR0, #5.5 ; MR0 = 5.0, ZF = 0, NF = 1 MMINF32 MR1, #2.5 ; MR1 = 2.5, ZF = 0, NF = 0 MMINF32 MR2, #-1.0 ; MR2 = -1.5, ZF = 0, NF = 1 MMINF32 MR2, #-1.5 ; MR2 = -1.5, ZF = 1, NF = 0								
See also	MMAXF32 M MMAXF32 M MMINF32 M	Ra, #16FH Ra, MRb Ra, MRb	li							



MMOV16 MARx, MRa, #16I Load the Auxiliary Register with MRa + 16-bit Immediate Value

operanus												
	MARx	Auxiliary re	egister MAF	R0 or MAR1								
	MRa	CLA Floati	ng-point re	gister (MR0 to	MR3)							
	#16I	16-bit imm	ediate valu	е								
Opcode	LSW: IIII MSW: 0111	IIII IIII IIII 1111 1101 00AA	(opcode	of MMOV16	MAR0, MRa	, #16I)						
	LSW: IIII MSW: 0111	IIII IIII IIII 1111 1111 00AA	(opcode	of MMOV16	MAR1, MRa	, #16I)						
Description	Load the a Refer to th MARx = MRa	auxiliary register, ne pipeline sectio a(15:0) + #16I;	MAR0 or n for imp	r MAR1, wi ortant infor	th MRa(18 mation reg	5:0) + 16-l garding th	oit immeo is instruc	diate val tion.	ue.			
Flags	This instru	uction does not m	odify flag	s in the Ms	STF regist	er:						
-	Flag TF ZF NF LUF						L	.VF				
	Modified	No N	0	No		No	١	lo				
Pipeline	This is a s phase of t will occur the auxilia • I1 and The tw occurs • I3 Loadin post-in registe addres with #_ • I4	single-cycle instru he pipeline. Any in the D2 phase ary registers: 12 to instructions foll . Thus these two ag of an auxiliary acrement address or or there will be ss-mode post incr _X. g with the 4th ins	ction. Th post incre of the pip owing MI instruction register c ing occur a conflict ement with truction N	e load of M ement of M eline. Ther MOV16 will ons will use occurs in th r in the D2 In the cas ill win and t	IAR0 or M AR0 or M efore the I use MAR the old van e EXE phase. Th se of a con the auxilia	IAR1 will of AR1 using following a R0/MAR1 I alue of M/ ase while ase while ase while ing I3 can offlict, the using ry register be the new	beccur in t g indirect applies w before th AR0 or M updates not use t update d r will not	he EXE address when loa e update IAR1. due to he auxili ue to be upda	sing ding e iary ated			
	MMOV	MMOVI16.										
	; Assume 1	; Assume MARO is 50, MRO is 10, and #_X is 20										
	MMOV16 <instruc <instruc <instruc <instruc <instruc< th=""><th>MAR0, MR0, #_X ction 1> ction 2> ction 3> ction 4> ction 5></th><th>; I1 Wil ; I2 Wil ; I3 Can ; I4 Wil ; I5</th><th>Load MAR0 1 use the 1 use the not use MA 1 use the</th><th>with addr old value old value RO new value</th><th>ess of X of MAR0 of MAR0 of MAR0</th><th>(20) + M (50) (50) (30)</th><th>R0 (10)</th><th></th></instruc<></instruc </instruc </instruc </instruc 	MAR0, MR0, #_X ction 1> ction 2> ction 3> ction 4> ction 5>	; I1 Wil ; I2 Wil ; I3 Can ; I4 Wil ; I5	Load MAR0 1 use the 1 use the not use MA 1 use the	with addr old value old value RO new value	ess of X of MAR0 of MAR0 of MAR0	(20) + M (50) (50) (30)	R0 (10)				
		Table 28. Pipeline Activity For MMOV16 MARx, MRa , #16I										
	Instruction	F1	F2	D1	D2	R1	R2	E	W			
	MMOV16 MA	.R0, MR0, #_X MMOV	16									
	11	l1	MMOV	16								
	12	12	11	MMOV16								
	13	13	12	11								
	14 15	14	13	12	11		MMOV46					
	GI	GI	14	13	12	11						
	16	16	15	14	13	12	11					

Texas Instruments

www.ti.com
Example 1

;	Calculate an offset into a sin/cos	s table
_	ClalTask1:	
	MMOV32 MR0,@_rad	; MRO = rad
	MMOV32 MRI,@_TABLE_SIZEDIVTwoPi	; MRI = TABLE_SIZE/(2*Pi)
	MMPYF32 MR1,MR0,MR1	; MR1 = rad* TABLE_SIZE/(2*Pi)
	MMOV32 MR2,@_TABLE_MASK	; MR2 = TABLE_MASK
	MF32TOI32 MR3,MR1	; MR3 = K=int(rad*TABLE_SIZE/(2*Pi))
	MAND32 MR3,MR3,MR2	; MR3 = K & TABLE_MASK
	MLSL32 MR3,#1	; MR3 = K * 2
	MMOV16 MAR0,MR3,#_Cos0	; MAR0 K*2+addr of table.Cos0
	MFRACF32 MR1,MR1	; 11
	MMOV32 MR0,@_TwoPiDivTABLE_SIZE	; 12
	MMPYF32 MR1,MR1,MR0	; I3
	MMOV32 MR0,@_Coef3	
	MMOV32 MR2,*MAR0[#-64]++	; MR2 = *MAR0, MAR0 += (-64)
	• • •	
	MSTOP ; end of task	



Instruction Set

Instruction Set				www.ti.com						
Example 2	; This task logs the last NUM_DATA_POINTS ; ADCRESULT1 values in the array VoltageCLA									
	; When the last element in the array has been ; filled, the task will go back to the ; the first element.									
	, ; Before starting the ADC conversions, force ; Task 8 to initialize the ConversionCount to zero									
	, _ClalTask2: MMOVZ16 MMOV16 MAR1, MUI16TOF32 MADDF32 MCMPF32 MF32TOUI16 MNOP MMOVZ16 MMOV16 MBCNDD MMOV1Z MNOP MNOP	<pre>MR0, @_ConversionCount MR0, #_VoltageCLA MR0, MR0 MR0, MR0, #1.0 MR0, #NUM_DATA_POINTS.0 MR0, MR0 MR2, @_AdcResult.ADCRESULT1 *MAR1, MR2 _RestartCount, GEQ MR1, #0.0</pre>	<pre>;I1 Current Conversion ;I2 Next array location ;I3 Convert count to float32 ;I4 Add 1 to conversion coun ;I5 Compare count to max ;I6 Convert count to Uint16 ;I7 Wait till I8 to read res ;I8 Read ADCRESULT1 ; Store ADCRESULT1 ; If count >= NUM_DATA_POINT ; Always executed: MR1=0</pre>	t ult S						
	MMOV16 MSTOP _RestartCount MMOV16	@_ConversionCount, MRU @_ConversionCount, MR1	<pre>; II branch not taken ; store current count ; If branch taken, restart c</pre>	ount						
	MSTOP ; This task ini ; to zero ; _ClalTask8: MMOVIZ MR0, MMOV16 @_Con MSTOP _ClaT8End:	tializes the ConversionCount #0.0 versionCount, MR0	; end of task							

See also

MMOV16 MARx, mem16 Load MAR1 with 16-bit Value

Operands

Operands	MARx	CLA auxili	arv register	MAR0 or MAR	1					
	mem16	16-bit dest	tination men	nory accessed	using indir	ect or direct a	addressi	ng modes	;	
Opcode	LSW: mmmm m MSW: 0111	nmmm mmmm mmmm 0110 0000 addr	(Opcode f	for MMOV16	MARO, men	m16)				
	LSW: mmmm m MSW: 0111	nmmm mmmm mmmm 0110 0100 addr	(Opcode f	for MMOV16	MAR1, men	m16)				
Description	Load MAR(section for	0 or MAR1 with important inform	the 16-bit nation reg	value point arding this i	ted to by instructio	mem16. R n.	tefer to	the pip	eline	
	MAR1 = [mei	n16];								
Flags	No flags M	STF flags are al	ffected.							
	Flag	TF 2	ZF	NF		LUF		LVF		
	Modified	No I	No	No		No		No		
	 pnase of th will occur ir the auxiliar I1 and I The two occurs. I3 Loading post-inc register address with #_X I4 	e pipeline. Any the D2 phase y registers: 2 instructions fol Thus these two of an auxiliary rement address or there will be -mode post incl K.	post incre of the pipe lowing MM instructio register o sing occur a conflict. rement wi	MOV16 will ns will use ccurs in the in the D2 p In the case I win snd th	use MAR the old va EXE phase. The of a cor he auxilia	ART USING following a 20/MAR1 b alue of MA ase while u as I3 canr offlict, the u ry register	andire pplies efore t .R0 or update oot use pdate will no	the upda MAR1. s due to the aux due to t be upo	ate kiliary	
	Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOV16. ; Assume MAR0 is 50 and @_X is 20									
	MMOV16 MAR <instruction <instruction <instruction <instruction< td=""><td colspan="9"><pre>MMOV16 MAR0, @_X ; Load MAR0 with the contents of X (20) <instruction 1=""> ; I1 Will use the old value of MAR0 (50) <instruction 2=""> ; I2 Will use the old value of MAR0 (50) <instruction 3=""> ; I3 Cannot use MAR0 <instruction 4=""> ; I4 Will use the new value of MAR0 (20) <instruction 5=""> ; I5</instruction></instruction></instruction></instruction></instruction></pre></td></instruction<></instruction </instruction </instruction 	<pre>MMOV16 MAR0, @_X ; Load MAR0 with the contents of X (20) <instruction 1=""> ; I1 Will use the old value of MAR0 (50) <instruction 2=""> ; I2 Will use the old value of MAR0 (50) <instruction 3=""> ; I3 Cannot use MAR0 <instruction 4=""> ; I4 Will use the new value of MAR0 (20) <instruction 5=""> ; I5</instruction></instruction></instruction></instruction></instruction></pre>								
		Table 29. Pipe	line Activ	ity For MN	10V16 M	AR0/MAR	1, me	m16		
	Instruction	F1	F2	D1	D2	R1	R2	Е	W	
	MMOV16 MAR	0, @_X MMOV	/16							
	11	11	MMOV	16						
	12	12	11	MMOV16						
	13	13	12	11	MMOV16					
	14	14	13	12	11	MMOV16				

15

16

MMOV16

11

MMOV1 6

11

12

14

15

13

14

12

13

15

16



Instruction Set

Instruction Set			www.ti.com								
Example	; This task logs the last NUM_DATA_POINTS ; ADCRESULT1 values in the array VoltageCLA										
	; ; When the las; ; filled, the ; ; the first ele	, ; When the last element in the array has been ; filled, the task will go back to the ; the first element.									
	; Before start: ; Task 8 to in: ;	ing the ADC conversions, forc itialize the ConversionCount	e to zero								
	, _ClalTask2: MMOVZ16 MUI16TOF32 MADDF32 MCMPF32 MF32TOUI16 MNOP MMOVZ16 MMOV16 MBCNDD MMOVIZ MNOP MNOP MNOP	MR0, @_ConversionCount MAR1, MR0, #_VoltageCLA MR0, MR0 MR0, MR0, #1.0 MR0, #NUM_DATA_POINTS.0 MR0, MR0 MR2, @_AdcResult.ADCRESULT1 *MAR1, MR2 _RestartCount, GEQ MR1, #0.0	<pre>;I1 Current Conversion ;I2 Next array location ;I3 Convert count to float32 ;I4 Add 1 to conversion count ;I5 Compare count to max ;I6 Convert count to Uint16 ;I7 Wait till I8 to read result ;I8 Read ADCRESULT1 ; Store ADCRESULT1 ; If count >= NUM_DATA_POINTS ; Always executed: MR1=0</pre>								
	_RestartCount MMOV16	@_ConversionCount, MR1	; If branch taken, restart count								
	MSTOP ; This task in. ; to zero ; _ClalTask8: MMOVIZ MMOVI6 MSTOP ClaT&End:	itializes the ConversionCount MR0, #0.0 @_ConversionCount, MR0	; end of task								

See also



MMOV16 mem16, MARx Move 16-bit Auxiliary Register Contents to Memory

	mem16	16	-bit destination me	mory accessed usin	ng indirect or direct a	ddressing modes				
	MARx	MARx CLA auxiliary register MAR0 or MAR1								
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm 0110 1000	mmmm (Opcode addr	for MMOV16 mem	16, MAR0)					
	LSW: mmmm MSW: 0111	mmmm mmmm 0110 1100	mmmm (Opcode addr	for MMOV16 mem	16, MAR1)					
Description	Store the mem16.	contents o	f MAR0 or MA	R1 in the 16-bit r	memory location	pointed to by				
	[mem16] =	MAR0;								
Flags	No flags N	/ISTF flags	are affected.							
	Flag	TF	ZF	NF	LUF	LVF				
	Modified	No	No	No	No	No				
Pipeline	This is a s	single-cycle	e instruction.							
Example										
See also										



MMOV16 mem16, MRa Move 16-bit Floating-Point Register Contents to Memory

Operatios											
	mem16	mem16 16-bit destination memory accessed using indirect or direct addressing modes									
	MRa	CLA floating-point sou	rce register (N	IR0 to MR3)							
Opcode	LSW: mmmm mmmm MSW: 0111 0101	mmmm mmmm 11aa addr									
Description	Move 16-bit value location pointed	ue from the lower 16 l to by mem16.	-bits of the	floating-point registe	r (MRa(15:0)) to the						
Flags	No flags MSTF	flags are affected.									
	Flag TF	ZF	NF	LUF	LVF						
	Modified No	No	No	No	No						
Pipeline	This is a single-	cycle instruction.									
Example	; This task log ; ADCRESULT1 va ; ; When the last ; filled, the t	gs the last NUM_DATA alues in the array V c element in the arr task will go back to	A_POINTS VoltageCLA ray has bee o the	n							
	<pre>, Before start: ; Task 8 to ini ; ClalTask2: MMOVZ16 MMOV16 MU116TOF32 MADDF32 MCMPF32 MF32TOUI16 MNOP MMOVI6 MBCNDD MMOV16 _RestartCount MMOV16 _RestartCount MMOV16 _STOP ; This task ini ; to zero ; ClalTask8: MMOVI2 MR0, MMOV16 @_Con MSTOP ClaT8End:</pre>	<pre>ing the ADC conversion MR0, @_Conversion MAR1, MR0, #_Volt MR0, MR0 MR0, MR0, #1.0 MR0, MR0, #1.0 MR0, MR0, MR0 MR2, @_AdcResult *MAR1, MR2 _RestartCount, GI MR1, #0.0 @_ConversionCount itializes the Conver #0.0 hversionCount, MR0</pre>	ions, force sionCount t nCount LageCLA DINTS.0 .ADCRESULT1 EQ 2, MR0 2, MR1 rsionCount	<pre>o zero ;I1 Current Conver ;I2 Next array loc ;I3 Convert count ;I4 Add 1 to conver ;I5 Compare count ;I6 Convert count ;I7 Wait till I8 t ;I8 Read ADCRESULT ; Store ADCRESULT ; If count >= NUM_; Always executed: ; If branch not ta ; store current co ; If branch taken, ; end of task</pre>	rsion to float32 ersion count to max to Uint16 to read result DATA_POINTS MR1=0 tken MSTOP ount restart count						
See also	MMOVIZ MRa, MMOVXI MRa,	#16FHiHex #16FLoHex									



MMOV32 mem32, MRa Move 32-bit Floating-Point Register Contents to Memory

Operands

Operands											
	MRa										
	mem32	mem32 32-bit destination memory accessed using indirect or direct addressing modes									
Opcode	LSW: mmmm mmm MSW: 0111 010	nm mmmm mmmm 00 11aa addr									
Description	[mem32] = MRa	Ra to 32-bit memory lo	cation indicate	d by mem32.							
Flags	This instruction	on modifies the followin	g flags in the N	ISTF register:							
	Flag	FF ZF	NF	LUF	LVF						
	Modified N	No No	No	No	No						
	No flags affect	cted.									
Pipeline	This is a singl	le-cycle instruction.									
Example	; Perform 5 m	nultiply and accumulat	e operations:								
	<pre>; 4th multipl ; 5th multipl ; Sth multipl ; Result = A ; _ClalTask1: MMOVI16 MNOP MNOP MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMACF32 MMOV32 MMOV32</pre>	<pre>Iy: D = X3 * Y3 Iy: E = X3 * Y3 Iy: E = X3 * Y3; + B + C + D + E MAR0, #_X MAR1, #_Y MR0, *MAR0[2]++ MR1, *MAR1[2]++ MR1, *MAR0[2]++ MR1, *MAR0[2]++ MR1, *MAR0[2]++ MR1, *MAR0[2]++ MR3, MR2, MR2, MF2 MR0, *MAR0[2]++ MR3, MR2, MR2, MF3 MR0, *MAR0[2]++ MR3, MR2, MR2, MF3 MR3 MR3 MR3 MR3 MR3 MR3 MR3 MR3 MR3 MR</pre>	; MAR0 ; MAR1 ; Dela ; Dela ; < ; MR0 ; < ; MR1 ; MR2 ; In p ; MR1 ; MR3 ; In p ; MR1 20, MR1 ; MR3 ; In p	points to X ar points to Y ar y for MAR0, MAR y for MAR0, MAR MAR0 valid = X0, MAR0 += 2 MAR1 valid = Y0, MAR1 += 2 = A = X0 * Y0 arallel MR0 = X = Y1, MAR1 += 2 = B = X1 * Y1 arallel MR0 = X = Y2, MAR2 += 2 = A + B, MR2 = arallel MR0 = X	ray ray 1 load 1 load 1, MARO += 2 2, MARO += 2 C = X2 * Y2 3						
	MMOV32 MMACF32 MMOV32 MMOV32 MMPYF32 MADDF32 MADDF32 MMOV32	MR1, *MAR1[2]++ MR3, MR2, MR2, MF MR0, *MAR0 MR1, *MAR1 MR2, MR0, MR1 MR3, MR3, MR2 MR3, MR3, MR2 @_Result, MR3	; MR1 ; MR1 ; In p ; MR1 ; MR1 ; MR2 ; in p ; MR3 ; Stor	= Y3 = (A + B) + C, arallel MR0 = X = Y4 = E = X4 * Y4 arallel MR3 = (= (A + B + C + e the result MS	MR2 = D = X3 * Y3 4 $A + B + C) + D$ $D) + E$ TOP ; end of task						

See also

MMOV32 mem32, MSTF



MMOV32 mem32, MSTF Move 32-bit MSTF Register to Memory

operanas							
	MSTF	flo	ating-point status r	egister			
	mem32	32	-bit destination me	mory			
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm 0111 0100	mmmm addr				
Description	Copy the (CLA's float	ing-point status	s register, MSTF	, to memory.		
	[mem32] =	MSTF;					
Flags	This instru	ction does	not modify flag	gs in the MSTF	register:		
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a s	ingle-cycle	instruction.				
Example							
See also	MMOV32	mem32. N	lRa				

MMOV32 MRa, mem32 {, CNDF} Conditional 32-bit Move

Operands MRa CLA floating-point destination register (MR0 to MR3) mem32 32-bit memory location accessed using direct or indirect addressing CNDF optional condition. Opcode LSW: mmmm mmmm mmmm MSW: 0111 00cn dfaa addr If the condition is true, then move the 32-bit value referenced by mem32 to the Description floating-point register indicated by MRa. if (CNDF == TRUE) MRa = [mem32]; CNDF is one of the following conditions: Encode ⁽¹⁾ CNDF **MSTF Flags Tested** Description 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 ZF == 1 OR NF == 1 Less than or equal to zero 1010 Test flag set TF == 1 TF 1011 NTF Test flag not set TF == 0 LUF == 1 1100 LU Latched underflow Latched overflow LVF == 1 1101 LV Unconditional 1110 UNC None UNCF⁽²⁾ Unconditional with flag 1111 None modification (1) Values not shown are reserved. (2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF 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 MSTF register: Flag TF ZF NF LUF LVF Modified Yes Yes No No No if(CNDF == UNCF) NF = MRa(31);

Pipeline

NF = MRa(31); ZF = 0; if(MRa(30:23) == 0) { ZF = 1; NF = 0; } } else No flags modified;

This is a single-cycle instruction.



Instruction Set

Example	; Given A, B, X, Ml and M2 are 32-bit floating-point ; numbers
	; if(A > B) calculate Y = X*M1 ; if(A < B) calculate Y = X*M2 ;
	; _ClalTask5: MMOV32 MR0, @_A MMOV32 MR1, @_B MCMPF32 MR0, MRB MMOV32 MR2, @_M1, EQ ; if A > B, MR2 = M1 ; Y = M1*X
	MMOV32 MR2, @_M2, NEQ ; if A < B, MR2 = M2 ; Y = M2*X
	MMOV32 MR3, @_X MMPYF32 MR3, MR2, MR3 ; Calculate Y MMOV32 @_Y, MR3 ; Store Y MSTOP ; end of task

See also

MMOV32 MRa, MRb {, CNDF} MMOVD32 MRa, mem32

MMOV32 MRa, MRb {, CNDF} Conditional 32-bit Move

Operands

	MRa	CLA floating-point destination register (MR0 to MR3)									
	MRb	CLA floating-	point source register (MR0 to MR3)								
	CNDF	optional cond	ition.								
Opcode	LSW: 0000 MSW: 0111	0000 cndf bbaa 1010 1100 0000									
Description	If the cond indicated b	lition is true, then m by MRa.	nove the 32-bit value in MRb	to the floating-point register							
	if (CNDF =	if (CNDF == TRUE) MRa = MRb;									
	CNDF is one of the following conditions:										
	Encode ⁽³⁾	CNDF	Description	MSTF 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 OR 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							
	4444	UNCE (4)	Unconditional with flag	None							

⁽⁴⁾ This is the default operation if no CNDF field is specified. This condition will allow the ZF, and NF 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 MSTF register:

			ig nage in ale h	ien regioten		
Flag	TF	ZF	NF	LUF	LVF	
Modified	No	Yes	Yes	No	No	
if(CNDF = {	= UNCF)					
NF = M	Ra(31); ZF	= 0;	-)			
if(MRa	(30:23) ==	$0) \{ ZF = 1; N \}$	F = 0;			

}
else No flags modified;

Pipeline

This is a single-cycle instruction.



Example

; Given: X =	8.0				
; Y =	7.0				
; A =	2.0				
; B =	5.0				
; _ClaTask1					
MMOV32	MR3,	@_X		;	MR3 = X = 8.0
MMOV32	MR0,	@_Y		;	MR0 = Y = 7.0
MMAXF32	MR3,	MR0		;	ZF = 0, $NF = 0$, $MR3 = 8.0$
MMOV32	MR1,	@_A,	GT	;	true, MR1 = $A = 2.0$
MMOV32	MR1,	@_B,	LT	;	false, does not load MR1
MMOV32	MR2,	MR1,	GT	;	true, MR2 = MR1 = 2.0
MMOV32	MR2,	MR0,	LT	;	false, does not load MR2
MSTOP					

See also

MMOV32 MRa, mem32{, CNDF}



MMOV32 MSTF, mem32 Move 32-bit Value from Memory to the MSTF Register

operanae									
	MSTF	CL	A status register						
	mem32	32-	bit source memory	y location					
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm 0111 0000	mmmm addr						
Description	Move from when nest MSTF = [me	n memory t ting function em32];	o the CLA's sta n calls (via MC	atus register MS CNDD).	TF. This instruct	ion is most useful			
Flags	This instru	iction modi	fies the followin	ng flags in the M	STF register:				
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	Yes	Yes	Yes	Yes	Yes			
	Loading the	Loading the status register will overwrite all flags and the RPC field. The MEALLOW field is not affected.							
Pipeline	i nis is a s	ingle-cycle	instruction.						
Example									
See also	MMOV32	mem32, M	STF						



MMOVD32 MRa, mem32 Move 32-bit Value from Memory with Data Copy

Operands

	MRa CLA floating-point register (MR0 to MR3)							
	mem32	mem32 32-bit memory location accessed using direct or indirect addressing						
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm m 0100 00aa a	ımmm ıddr					
Description	Move the 32-bit value referenced by mem32 to the floating-point register indicated by MRa.							
	MRa = [men [mem32+2]	n32]; = [mem32];						
Flags	This instruction modifies the following flags in the MSTF register:							
	Flag	TF	ZF	NF	LUF			
	NF = MRa(3 ZF = 0; if(MRa(30)	31); :23) == 0){	ZF = 1; NF	= 0; }				
Pipeline	This is a single-cycle instruction.							
Example	<pre>; sum = X(; ; X1 ; X1 ; Y1 ; [ClalTask2 MMOV32 MMOV32 MMOV32 MMOV32 ; MR3 = X1 ; MR0 = A2 MMACF3 MMOV32 ; MR3 = X(; MR0 = A1 MMACF3 MMOV32 ; MR3 = X(; MR0 = A1 MMACF3 MMOV32 ; MR3 = X(; MR0 = A1 MMACF3 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMO</pre>	<pre>D*B0 + X1*B1 2 = X1 L = X0 2 = Y1 L = sum 2: 2 MR0, @_B2 2 MR1, @_X2 32 MR2, MR1, 2 MR0, @_B1 32 MR1, @_X1 32 MR3, MR1, 2 MR0, @_B0 32 MR1, @_X0 L*B1 + X2*B2 32 MR3, MR2, 2 MR0, @_A2 2 MR1, @_Y2 D*B0 + X1*B1 32 MR3, MR2, 2 MR0, @_A1 32 MR3, MR2, 32 MR3, MR3, 33 MR3, 34 MR3, MR3, 35 MR3, MR3, 35 MR3, MR3, 36 MR3, MR3, 37 MR3, MR3, 37 MR3, MR3, 38 MR3, MR3, 39 MR3, MR3, 30 MR3, MR3, 30 MR3, MR3, 31 MR3, 32 MR3, MR3, 33 MR3, 34 MR3, MR3, 35 MR3, MR3, 36 MR3, MR3, 37 MR3, MR3, 37 MR3, MR3, 38 MR3, MR3, 39 MR3, MR3, 30 MR3, MR3, 30 MR3, MR3, 31 MR3, 32 MR3, MR3, 33 MR3, 34 MR3, MR3, 35 MR3, MR3, 35 MR3, MR3, 36 MR3, MR3, 37 MR3, MR3, 37 MR3, MR3, 38 MR3, MR3, 39 MR3, MR3, 30 MR3, MR3, 30 MR3, MR3, 31 MR3, 32 MR3, MR3, 33 MR3, 34 MR3, MR3, 35 MR3, MR3, 35 MR3, MR3, 36 MR3, MR3, 37 MR3, MR3, 37 MR3, MR3, 38 MR3, MR3, 39 MR3, MR3, 30 MR3, MR3, 30 MR3, MR3, 31 MR3, 32 MR3, MR3, 32 MR3, MR3, 33 MR3, 34 MR3, MR3, 35 MR3, MR3, 35 MR3, MR3, 35 MR3, MR3, 36 MR3, MR3, 37 MR3, MR3, 37 MR3, MR3, 38 MR3, MR3, 39 MR3, MR3, 30 MR3, MR3, 30 MR3, MR3, 30 MR3, MR3, 31 MR3, 31 MR3, 32 MR3, MR3, 31 MR3, 31</pre>	; MR0 ; MR0 ; MR1 MR0 ; MR2 ; MR0 ; MR1 MR0 ; MR3 ; MR0 ; MR1 MR2 ; MR1, ; MR1 MR2 ; MR1, ; MR1 MR2 ; MR3 MR0 ; MR2 MR2 ; MR3 MR0 ; MR2 ; Y1 ; end	<pre>Y1*A1 + Y2*B2 = B2 = X2 = X2*B2 = B1 = X1, X2 = X1 = X1*B1 = B0 = X0, X1 = X0 B0 MR0 = Y2 R2 = Y2*A2 MR0 = Y1, Y2 = Y1 = Y2*A2 + X0*B0 = Y1*A1 = Y1*A1 + Y2*A2 = MR3 of task</pre>	+ X1*B1 + X2*E + X0*B0 + X1*E	32 31 + X2*B2		

See also

MMOV32 MRa, mem32 {,CNDF}


MMOVF32 MRa, #32F Load the 32-bits of a 32-bit Floating-Point Register

Operands	This instruction is an alias for MMOVIZ and MMOVXI instructions. The second operand is translated by the assembler such that the instruction becomes:							
	MRa CLA floating-point destination register (MR0 to MR3)							
	#32F	imme	ediate float value	e represented in floati	ing-point representa	ation		
Opcode	LSW: IIII IIII IIII IIII (opcode of MMOVIZ MRa, #16FHiHex) MSW: 0111 1000 0100 00aa LSW: IIII IIII IIII IIII (opcode of MMOVXI MRa, #16FLoHex) MSW: 0111 1000 1000 00aa							
Description	Note: This i representati point format	nstruction a ion. To spe i) use the N	accepts the ir cify the imme IOVI32 MRa	nmediate operar ediate value as a , #32FHex instru	nd only in floatin hex value (IEE ction.	ng-point E 32-bit floating-		
	Load the 32-bits of MRa with the immediate float value represented by #32F.							
	#32F is a flo accept a flo represented MRa = #32F;	oat value re at value re I as #3.0. #	epresented in presented in 0x40400000	floating-point re floating-point rep will result in an e	presentation. Th presentation. Th error.	he assembler will o hat is, 3.0 can only b	nly ce	
Flags	This instruction modifies the following flags in the MSTF register:							
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No		
Pipeline	Depending of the IEEE convert MM floating-poir into MMOVI	on #32FH, 32-bit float OVF32 into t format of Z and MM	this instruction ting-point form to only MMOV #32F are no OVXI instruct	on takes one or t nat of #32F are z /IZ instruction. If t zeros, then the ions.	wo cycles. If all zeros, then the the lower 16-bit assembler will	of the lower 16-bits assembler will ts of the IEEE 32-bit convert MMOVF32	3 it	
Example	MMOVF32 MR1	, #3.0	; MR1 = 3.0 ; Assembler ; MMOVIZ MR1	(0x40400000) converts this i , #0x4040	nstruction as			
	MMOVF32 MR2	, #0.0	; MR2 = 0.0 ; Assembler ; MMOVIZ MR2	(0x00000000) converts this i 2, #0x0	nstruction as			
	MMOVF32 MR3	, #12.265	; MR3 = 12.6 ; Assembler ; MMOVIZ MR3 ; MMOVXI MR3	25 (0x41443D71) converts this i 3, #0x4144 3, #0x3D71	nstruction as			
See also	MMOVIZ M MMOVXI M MMOVI32 N	Ra, #16FH Ra, #16FL //Ra, #32FI	i oHex Hex					

MMOVI16 MARx, #16I Load the Auxiliary Register with the 16-bit Immediate Value

Operands

peralius									
	MARx	Auxiliary re	gister MAR	0 or MAR1					
	#16I	16-bit imme	ediate value	e					
pcode	LSW: IIII III MSW: 0111 111	I IIII IIII 1 1100 0000	(opcode d	of MMOVI16	5 MAR0, #1	61)			
	LSW: IIII III MSW: 0111 111	I IIII IIII 1 1110 0000	(opcode d	of MMOVI16	5 MAR1, #1	6I)			
escription	Load the auxil pipeline sectio MARx = #161;	liary register, l on for importa	MAR0 or nt informa	MAR1, w ation rega	ith a 16-bit rding this i	t immediate v nstruction.	/alue. Refei	to the	
lags	This instructio	n does not me	odify flag	s in the M	STF regist	er:			
	Flag 1	TF ZF		NF		LUF	LVF		
	Modified N	No No	D	No		No	No		
	 when loading I1 and I2 The two in occurs. Th I3 Loading of post-increar register or address-m with #_X. 	 when loading the auxiliary registers: I1 and I2 The two instructions following MMOVI16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1. I3 Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win snd the auxiliary register will not be updated with # X.							
	 I4 Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOVI16. Assume MAR0 is 50 and #_X is 20 								
	MMOVI16 MAR0, <instruction <instruction <instruction <instruction< td=""><td>#_X 1> ; 2> ; 3> ; 4> ; 5> ;</td><td>; Lo Il Will I2 Will I3 Canno I4 Will I5</td><td>bad MARO v use the c use the c bt use MAF use the r</td><td>vith addre old value old value RO new value</td><td>ss of X (20) of MAR0 (50) of MAR0 (50) of MAR0 (20)</td><td></td><td></td></instruction<></instruction </instruction </instruction 	#_X 1> ; 2> ; 3> ; 4> ; 5> ;	; Lo Il Will I2 Will I3 Canno I4 Will I5	bad MARO v use the c use the c bt use MAF use the r	vith addre old value old value RO new value	ss of X (20) of MAR0 (50) of MAR0 (50) of MAR0 (20)			
	٦	Table 30. Pip	eline Act	tivity For	MMOVI16	MAR0/MAR	1, #16l		
	Instruction	F1	F2	D1	D2	R1 R2	2 E	w	
	MMOVI16 MAR0, #	#_X MMOVI	16						
	11	11	MMOVI	16					
	12	12	11	MMOVI1	6				
	13	13	12	11	MMOVI16				
	14	14	13	12	11	MMOVI16			

11

12

MMOVI16

11

MMOVI 16

12

13

14

15

13

14

15

16

15

16



MMOVI32 MRa, #32FHex Load the 32-bits of a 32-bit Floating-Point Register with the immediate

		n							
	#32FHex A 32-bit immediate value that represents an IEEE 32-bit floating-point value.								
	This instruction is an alias for MMOVIZ and MMOVXI instructions. The second operand is translated by the assembler such that the instruction becomes:								
	MMOVIZ MRa, #16FI MMOVXI MRa, #16FI	liHex JoHex							
Opcode	LSW: IIII IIII II MSW: 0111 1000 01	II IIII (opcoo 00 00aa	de of MMOVIZ MRa,	#16FHiHex)					
	LSW: IIII IIII II MSW: 0111 1000 10	III IIII (opcoo 000 00aa	de of MMOVXI MRa,	#16FLoHex)					
Description	Note: This instruct immediate value v instruction.	ion only accept vith a floating-p	ts a hex value as t oint representatior	he immediate op use the MMOVF	erand. To specify the F32 MRa, #32F				
	Load the 32-bits o	f MRa with the	immediate 32-bit I	hex value represe	ented by #32Fhex.				
	#32Fhex is a 32-b value of a floating That is, 3.0 can or	it immediate he -point number. nly be represen	ex value that repre The assembler wil ted as #0x404000	sents the IEEE 3 I only accept a he 00. #3.0 will resu	2-bit floating-point ex immediate value. It in an error.				
Flags	This instruction m	odifies the follo	wing flags in the N	ISTF register:					
	Flag TF	ZF	NF	LUF	LVF				
	Modified No	NO	NO	NO	NO				
Pipeline	Depending on #32 16-bits of #32FHe instruction. If the I MOVI32 to a MMC	PFHex, this inst x are zeros, the ower 16-bits of DVIZ and a MM	ruction takes one of an assembler will of #32FHex are not a OVXI instruction.	or two cycles. If a convert MOVI32 t zeros, then asser	II of the lower o the MMOVIZ mbler will convert				
Example	MOVI32 MR1, #0>	:40400000 ; MRI ; Ass ; MMC	= 0x40400000 sembler converts f DVIZ MR1, #0x4040	this instruction	as				
	MOVI32 MR2, #0x0	00000000 ; MR2 ; Ass ; MM0	2 = 0x00000000 sembler converts f DVIZ MR2, #0x0	this instruction	as				
	MOVI32 MR3, #0x4	0004001 ; MR3 ; Ass ; MM0 ; MM0	B = 0x40004001 sembler converts DVIZ MR3, #0x4000 DVXI MR3, #0x4001	this instruction	as				
	MOVI32 MR0, #0x0	00004040 ; MR(; Ass ; MM(; MM() = 0x00004040 sembler converts f DVIZ MR0, #0x0000 DVXI MR0, #0x4040	this instruction	as				
See also	MMOVIZ MRa, #1 MMOVXI MRa, #1 MMOVF32 MRa, ;	6FHi 6FLoHex #32F							

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

operanas				(
	MRa	flo	pating-point registe	r (MR0 to MR3)				
	#16FHi	A flo	16-bit immediate v ating-point value.	alue that represents The low 16-bits of th	s the upper 16-bits of ne mantissa are assu	an IEEE 32-bit med to be all 0.		
Opcode	LSW: IIII MSW: 0111	IIII IIII 1000 0100	IIII 00aa					
Description	Load the u 16-bits of I	ipper 16-b VRa.	its of MRa with	the immediate	value #16FHi an	d clear the low		
	#16FHiHe: 32-bit float assembler represente	x is a 16-b ing-point will only a ed as #-1.5	bit immediate v value. The low accept a decim 5 or #0xBFC0.	alue that represe 16-bits of the m al or hex immed	ents the upper 16 antissa are assu liate value. That	δ-bits of an IEEE med to be all 0. The is, -1.5 can be	Э	
	By itself, M the lowest (0x408000 32-bits of a MMOVXI i	By itself, MMOVIZ is useful for loading a floating-point register with a constant in which 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). If a constant requires all 32-bits of a floating-point register to be iniitalized, then use MMOVIZ along with the MMOVXI instruction						
	MRa(31:16) MRa(15:0)	= #16FHi = 0;	.;					
Flags	This instruction modifies the following flags in the MSTF register:							
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No		
Pipeline	This is a s	ingle-cycle	e instruction.					
Example	; Load MRC MMOVI MMOVI	<pre>; Load MR0 and MR1 with -1.5 (0xBFC00000) MMOVIZ MR0, #0xBFC0 ; MR0 = 0xBFC00000 (1.5) MMOVIZ MR1, #-1.5 ; MR0 = -1.5 (0xBFC00000)</pre>						
	; Load MR2 MMOVI MMOVX	with pi Z MR2, XI MR2,	= 3.141593 (0) #0x4049 #0x0FDB	x40490FDB) ; MR0 = 0x40490 ; MR0 = 0x40490	000 FDB			
See also	MMOVF32 MMOVI32 MMOVXI I	2 MRa, #3 MRa, #32 MRa, #16I	2F 2FHex ⁻ LoHex					



MMOVZ16 MRa, mem16 Load MRx with 16-bit Value

-	MRa	CL	A floating-point de	stination register (M	R0 to MR3)				
	mem16	16	-bit source memory	/ location					
Opcode	LSW: mmmm MSW: 0111	mmmm mmmm 0101 10aa	addr						
Description	Move the MRa. MRa(31:16	16-bit valu	e referenced by	/ mem16 to the	floating-point rec	gister indicated by			
	MRa(15:0)	= [mem16]	;						
Flags	This instru	iction mod	ifies the followir	ng flags in the M	ISTF register:				
	Flag		ZF	NF	LUF				
	Modified	NO	Yes	Yes	NO	NO			
	The MST	The MSTF register flags are modified based on the integer results of the operation.							
	NF = 0; if (MRa(3)	1:0)== 0)	{ ZF = 1; }		-				
Pipeline	This is a s	ingle-cycle	e instruction.						

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

	MRa	CL	A floating-point re	gister (MR0 to MR3)					
	#16FLoHex	A flo	16-bit immediate h ating-point value.	ex value that represe The upper 16-bits wil	ents the lower 16-bits Il not be modified.	s of an IEEE 32-bit			
Opcode	LSW: IIII MSW: 0111	IIII IIII 1000 1000	IIII 00aa						
Description	Load the lo represents MRa will n initialize al ^{MRa(15:0)}	Load the low 16-bits of MRa with the immediate value #16FLoHex. #16FLoHex represents the lower 16-bits of an IEEE 32-bit floating-point value. The upper 16-bits of MRa will not be modified. MMOVXI can be combined with the MMOVIZ instruction to initialize all 32-bits of a MRa register.							
	MRa(31:16)	= Unchan	iged ;						
Flags									
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	No	No	No	No			
Pipeline	This is a si	ngle-cycle	e instruction.						
Example	; Load MRO MMOVIZ MMOVXI	with pi MRO MRO	= 3.141593 (02 ,#0x4049 ; M ,#0x0FDB ; M	x40490FDB) MR0 = 0x40490000 MR0 = 0x40490FDE) 3				
See also	MMOVIZ N	/IRa, #16F	FHi						

MMPYF32 MRa, MRb, MRc 32-bit Floating-Point Multiply

Operands						
	MRa	CL	A floating-poir	t destination registe	r (MR0 to MR3)	
	MRb	CL	A floating-poir	t source register (M	R0 to MR3)	
	MRc	CL	A floating-poir	t source register (M	R0 to MR3)	
Opcode	LSW: 0000 (MSW: 0111]	0000 00cc 100 0000	bbaa 0000			
Description	Multiply the MRa = MRb *	contents	of two float	ing-point registe	rs.	
Flags	This instruc	tion modif	ies the follo	owing flags in the	e MSTF register:	
	Flag	TF	ZF	NF	LUF	LVF
	Modified	No	No	No	Yes	Yes
	 LUF = 1 LVF = 1 	if MMPY	F32 genera F32 genera F32 genera	tes an underflow tes an overflow	v condition. condition.	
Pipeline	This is a sir	ngle-cycle	instruction			
Example	<pre>; Calculate ; Ye = Esti ; Ye = Ye*(; Ye = Ye*(; _ClalTask1: _MMOV32 _MEINVF3_ _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMOV32 _MMPYF32 _MMOV32 _MMOV32 _MSTOP</pre>	<pre>Num/Den Imate(1/X) 2.0 - Ye³ 2.0 - Ye³ MR1, 32 MR2, 2 MR3, 2 MR2, 2 MR3, 2 MR</pre>	<pre>using a Ne (X) (%_Den MR1 MR2, MR1 #2.0, MR3 MR2, MR1 (%_Num #2.0, MR3 MR2, MR3 (%_Den MR0, EQ MR2, MR0 St, MR0</pre>	<pre>% Wton-Raphson al</pre>	gorithum for 1/Den Estimate(1/Den) Ye*Den Ye*(2.0 - Ye*Den) Ye*Den Ye*(2.0 - Ye*Den) To Set Sign 0) Change Sign Of *Num	Num
See also	MMPYF32 MMPYF32 MMPYF32 MMPYF32 MMPYF32 MMACF32	MRa, #16 MRa, MRI MRd, MR0 MRd, MR0 MRa, MR1 MR3, MR	FHi, MRb b, MRc M e, MRf M e, MRf M b, MRc N 2, MRd, Mf	IADDF32 MRd, I MOV32 MRa, m MOV32 mem32, ISUBF32 MRd, I Re, MRf MMO\	MRe, MRf em32 MRa MRe, MRf √32 MRa, mem32	



MMPYF32 MRa, #16FHi, MRb 32-bit Floating-Point Multiply

Operanus									
	MRa	CL	A floating-point de	estination register (M	R0 to MR3)				
	#16FHi	#16FHi 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.							
	MRc	CL	A floating-point so	ource register (MR0 t	o MR3)				
Opcode	LSW: IIII MSW: 0111	IIII IIII 0111 1000	IIII baaa						
Description	Multiply MF the result o	२b with the	e floating-point tion in MRa.	value represent	ed by the immed	liate operand. Sto	re		
	#16FHi is a floating-poi most usefu Some exar (0xBFC000 That is, the MRa = MRb This instruc	 a 16-bit im nt value. l for repre nples are 000). The e value -1. * #16FHi: ction can a 	mediate value The low 16-bits senting consta 2.0 (0x400000 assembler will 5 can be repre 0; also be written	that represents of the mantissa ints where the lo 00), 4.0 (0x4080 accept either a l sented as #-1.5 as MMPYF32 M	the upper 16-bits a are assumed to west 16-bits of to 00000), 0.5 (0x3F nex or float as th or #0xBFC0. IRa, MRb, #16FF	s of an IEEE 32-b be all 0. #16FHi ne mantissa are 0 -000000), and -1. e immediate value	it is). 5 e.		
Flags	This instruc	ction modi	fies the followi	ng flags in the N	ISTF register:.				
	Flag			NF	LUF				
	The MSTF LUF = 1 LVF = 1 	 The MSTF register flags are modified as follows: LUF = 1 if MMPYF32 generates an underflow condition. LVF = 1 if MMPYF32 generates an overflow condition. 							
Pipeline	This is a si	ngle-cycle	instruction.						
Example 1	; Same as MMOVIZ MMPYF3 MMOV32	example 2 MR3, 2 MR0, @_X,	but #16FHi is #2.0 ; MF #3.0, MR3 ; MF MR0 ; Sa	s represented in R3 = 2.0 (0x4000 R0 = 3.0 * MR3 = ave the result i	n float 00000) = 6.0 (0x40C0000 in variable X	10)			
Example 2	; Same as MMOVIZ MMPYF3 MMOV32	example 1 MR3, 2 MR0, @_X,	but #16FHi is #2.0 #0x4040, MR3 MR0	s represented in F MR3 = 2.0 (0x4 F MR0 = 0x4040 F Save the result	n Hex 40000000) * MR3 = 6.0 (0x4 Lt in variable 3	10C00000)			



Example 3	<pre>; Given X, M and B are IQ24 numbers: ; X = IQ24(+2.5) = 0x02800000 ; M = IQ24(+1.5) = 0x01800000 ; B = IQ24(-0.5) = 0xFF800000 ; ; Calculate Y = X * M + B ; ; ClalTask2: ; ClalTask2: ; ; Convert M, X and B from IQ24 to float MI32TOF32 MR0, @_M ; MR0 = 0x4BC00000 MI32TOF32 MR1, @_X ; MR1 = 0x4C200000 MI32TOF32 MR2, @_B ; MR2 = 0xCB000000 MMPYF32 MR2, @_B ; MR2 = 0xCB000000 MMPYF32 MR1, MR1, #0x3380 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000) MMPYF32 MR1, MR1, #0x3380 ; X = 1/(1*2^24) * iqm = 1.5 (0x40200000) MMPYF32 MR2, MR2, #0x3380 ; X = 1/(1*2^24) * iqb =5 (0x40200000) MMPYF32 MR2, MR2, MR3 ; Y=MX+B = 3.25 (0x40500000) ; Convert Y from float32 to IQ24 MMPYF32 MR2, MR2, #0x4B80 ; Y * 1*2^24 MF32TOI32 MR2, MR2, #0x4B80 ; Y * 1*2^24 MF32TOI32 MR2, MR2, #0x4B80 ; Y * 1*2^24 MMPYF32 MR2, MR2, WR2, #0x4B80 ; Y * 1*2^24 MR32TOI32 MR2, MR2, WR2, WR2 ; IQ24(Y) = 0x03400000 MMOV32 @_Y, MR2 ; end of task</pre>
See also	MMPYF32 MRa, MRb, #16FHi MMPYF32 MRa, MRb, MRc MMPYF32 MRa, MRb, MRc MADDF32 MRd, MRe, MRf



MMPYF32 MRa, MRb, #16FHi 32-bit Floating-Point Multiply

Operands							
	MRa	CL	A floating-point de	stination register (M	R0 to MR3)		
	MRb	CL	A floating-point sc	urce register (MR0 t	o MR3)		
	#16FHi	A 1 floa	16-bit immediate v ating-point value.	alue that represents The low 16-bits of the	the upper 16-bits of e mantissa are assur	an IEEE 32-bit ned to be all 0.	
Opcode	LSW: IIII : MSW: 0111 (IIII IIII 0111 1000	IIII baaa				
Description	Multiply MF the result o	₹b with the of the addi	e floating-point tion in MRa.	value represent	ed by the immed	liate operand. Sto	ore
	#16FHi is a floating-poi most usefu Some exan (0xBFC000 That is, the MRa = MRb	16-bit im nt value. I for repre nples are 000). The value -1. * #16FH1:	mediate value The low 16-bits senting consta 2.0 (0x400000 assembler will 5 can be repre	that represents of the mantissants where the lo 00), 4.0 (0x4080 accept either a h sented as #-1.5	the upper 16-bits a are assumed to west 16-bits of th 00000), 0.5 (0x3F nex or float as th or #0xBFC0.	s of an IEEE 32-b be all 0. #16FHi ne mantissa are 0 ⁷ 000000), and -1. e immediate valu	it is). 5 e.
Flags	This instruc	tion can a	fies the followi	ng flags in the M	ISTF register:.).	
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	Yes	Yes	
	The MSTF LUF = 1 LVF = 1 	register fl if MMPY if MMPY	ags are modifi F32 generates F32 generates	ed as follows: an underflow co an overflow cor	ondition. Idition.		
Pipeline	This is a sir	ngle-cycle	instruction.				
Example 1	;Same as e: MMOVIZ MMPYF3: MMOV32	xample 2 1 MR3, 2 2 MR0, 1 @_X, 2	but #16FHi is #2.0 ; MR3, #3.0 ; MR0 ; S	represented in IR3 = 2.0 (0x400 IR0 = MR3 * 3.0 Gave the result	float 000000) = 6.0 (0x40C000 in variable X	00)	
Example 2	;Same as al MMOVIZ MMPYF3: MMOV32	Dove exam MR3, = 2 MR0, 1 @_X, =	ple but #16FH: #2.0 MR3, #0x4040 MR0	is represented ; MR3 = 2.0 (02 ; MR0 = MR3 * (; Save the resu	d in Hex <40000000))x4040 = 6.0 (0x ult in variable	:40C00000) X	



Example 3	<pre>; Given X, M and B are IQ24 numbers: ; X = IQ24(+2.5) = 0x02800000 ; M = IQ24(+1.5) = 0x01800000 ; B = IQ24(-0.5) = 0xFF800000 ; ; Calculate Y = X * M + B ; .ClalTask2: ;</pre>
	; Convert M, X and B from IQ24 to float
	$MIS210F32 MR0, @_M , MR0 = 0x4500000 MIS210F32 MP1 @ X : MP1 = 0x40200000$
	MI32TOF32 MR2, @ B ; MR2 = 0xCB000000
	MMPYF32 MR0, $\#0x3380$, MR0 ; M = $1/(1*2^{24})$ * igm = 1.5 (0x3FC00000)
	MMPYF32 MR1, $\#0x3380$, MR1; X = $1/(1*2^{2}4) * iqx = 2.5$ (0x40200000)
	MMPYF32 MR2, $\#0x3380$, MR2; $B = 1/(1*2^{2}4) * iqb =5$ (0xBF000000)
	MMPYF32 MR3, MR0, MR1 ; M*X
	MADDF32 MR2, MR2, MR3 ; Y=MX+B = 3.25 (0x40500000)
	; Convert Y from float32 to IQ24
	MMPYF32 MR2, #0x4B80, MR2 ; Y * 1*2^24
	MF32TOI32 MR2, MR2 ; $IQ24(Y) = 0x03400000$
	MMOV32 @_Y, MR2 ; store result
	MSTOP ; end of task
See also	MMPYF32 MRa, #16FHi, MRb MMPYF32 MRa, MRb, MRc



MMPYF32 MRa, MRb, MRc||MADDF32 MRd, MRe, MRf 32-bit Floating-Point Multiply with Parallel Add

Operands									
-	MRa CLA floating-point destination register for MMPYF32 (MR0 to MR3) MRa cannot be the same register as MRd								
	MRb CLA floating-point source register for MMPYF32 (MR0 to MR3)								
	MRc	CL	A floating-point so	ource register for MN	IPYF32 (MR0 to MR3	3)			
	MRd	CL MF	A floating-point de	estination register for ame register as MR	MADDF32 (MR0 to a	MR3)			
	MRe	CL	A floating-point so	ource register for MA	DDF32 (MR0 to MR3	3)			
	MRf	CL	A floating-point so	ource register for MA	DDF32 (MR0 to MR3	3)			
Opcode	LSW: 0000 MSW: 0111	LSW: 0000 ffee ddcc bbaa MSW: 0111 1010 0000 0000							
Description	Multiply the MRa = MRb MRd = MRe	Multiply the contents of two floating-point registers with parallel addition of two registers. MRa = MRb * MRc; MRd = MRe + MRf;							
Restrictions	The destin MRa cann	ation regis ot be the s	ster for the MM same register a	PYF32 and the as MRd.	MADDF32 must	be unique. That is,			
Flags	This instru	ction modi	fies the followi	ng flags in the M	ISTF register:				
	Flag	TF	ZF	NF	LUF	LVF			
	Modified	No	No	No	Yes	Yes			
	The MSTF LUF = LVF =	register fl 1 if MMPY 1 if MMPY	ags are modifi F32 or MADDI F32 or MADDI	ed as follows: F32 generates a F32 generates a	n underflow condi	dition. tion.			
Pipeline	Both MMP	YF32 and	MADDF32 cor	mplete in a singl	e cycle.				



www	ti.	com	
** ** **	. u.	00111	

<pre> X and Y are 32-bit floating point arrays i Ist multiply: A = X0 * Y0 Zond multiply: B = X1 * Y1 i 3rd multiply: C = X2 * Y2 i 4 th multiply: D = X3 * Y3 i the multiply: A = X0 + D + E</pre>	Example	; P	erform 5 mul	ltiply	and accumulate ope	era	ations:
<pre>// ist multiply: A = X0 * Y0 // 2nd multiply: B = X1 * Y1 // 3rd multiply: C = X2 * Y2 // 4th multiply: D = X3 * Y3 // Result = A + B + C + D + E // ClaITask1:</pre>		; x	and Y are 3	32-bit	floating point ar:	ra	ys
<pre>/ is: Multiply: A = X0 i0 ; 2nd multiply: C = X2 * Y2 ; 4th multiply: D = X3 * Y3 ; 5th multiply: E = X3 * Y3 ; ; Result = A + B + C + D + E ; </pre>		;	at multiplu	• 7 -	V0 * V0		
<pre>i 3rd multiply: C = X2 * Y2 i 3rd multiply: C = X2 * Y2 i 4th multiply: D = X3 * Y3 i 5th multiply: E = X3 * Y3 i claitast: MMOVI16 MAR0, #_X</pre>		; 2	nd multiply	· A =	X0 " 10 X1 * V1		
<pre> #th multiply: D = X3 * Y3 ; Sth multiply: E = X3 * Y3 ; sth multiply: E = X3 * Y3 ; Result = A + B + C + D + E ; </pre>		; 3:	rd multiply	: C =	x2 * y2		
<pre>; 5th multiply: E = X3 * Y3 ; Result = A + B + C + D + E ;</pre>		; 4	th multiply	: D =	x3 * y3		
<pre>; ; Result = A + B + C + D + E ; .ClalTask1: MMOVI16 MAR0, #_X ; MAR0 points to X array MMOVI16 MAR1, #_Y ; MAR1 points to Y array MNOP ; Delay for MAR0, MAR1 load NNOP ; Delay for MAR0, MAR1 load Y = Y = Y = Y = Y = Y = Y = Y = Y =</pre>		; 5	th multiply	: E =	X3 * Y3		
<pre>; Result = A + B + C + D + E ;</pre>		;					
<pre>; _ClalTask1: _MMOV116 MAR0, #_X ; MAR0 points to X array MMOV16 MAR1, #_Y ; MAR1 points to Y array MMOP ; Delay for MAR0, MAR1 load ; C = MAR0 valid MMOV32 MR0, *MAR0[2]++ ; MR0 = X0, MAR0 += 2 ; C = MAR1 valid MMOV32 MR1, *MAR1[2]++ ; MR1 = Y0, MAR1 += 2 MMPYF32 MR2, MR0, MR1 ; MR2 = A = X0 * Y0 MMOV32 MR1, *MAR0[2]++ ; In parallel MR0 = X1, MAR0 += 2 MM0V32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR1, *MAR1[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3, MAR2 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, HP2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, HP2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, HP2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, HP2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, HP2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C + D + D MADF32 MR3, MR3, MR2 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR3 = (A + B + C) + D MADF32 MR3, MR3, MR3 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MADF32 MR3, MR3, MR3, MR3 ; Store the result MADF34 wAR1 ; MR3 ; Store the result</pre>		; R	esult = A +	B + C	2 + D + E		
		;					
<pre>MMOVI16 MAR0, #_X ; MARD points to X array MMOVI16 MAR1, #_Y ; MARL points to Y array MNOP ; Delay for MAR0, MAR1 load ; C MAR0 valid MMOV32 MR0, *MAR0[2]++ ; MR1 = Y0, MAR0 += 2 ; < MAR1 valid MMOV32 MR1, *MAR1[2]++ ; MR1 = Y0, MAR1 += 2 MMPYF32 MR2, MR0, MR1 ; MR2 = A = X0 * Y0 [] MMOV32 MR1, *MAR0[2]++ ; In parallel MR0 = X1, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 [] MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 [] MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 [] MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 [] MMOV32 MR1, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 [] MMOV32 MR1, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 [] MMOV32 MR1, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR3 = (A + B + C) + D MADF32 MR3, MR3, MR2, MR3 ; Store the result MADF32 MR3, MR3, MR3, MR3 ; Store the result MSTOP ; end of task</pre>		_Cla	alTask1:				
<pre>MMOVILE MARI, #_Y ; MARI points to Y array MNOP ; Delay for MARO, MARI load ; < MARO valid MMOV32 MRO, *MARO[2]++ ; MRO = XO, MARO += 2 ; < MARI valid MMOV32 MR1, *MARI[2]++ ; MRI = YO, MARI += 2 MMPYF32 MR2, MRO, MRI ; MR2 = A = XO * YO MMOV32 MRO, *MARO[2]++ ; In parallel MRO = X1, MARO += 2 MMPYF32 MR3, MRO, MRI ; MR3 = B = X1 * Y1 MMOV32 MRO, *MARO[2]++ ; In parallel MRO = X2, MARO += 2 MMPYF32 MR3, MRO, MRI ; MR3 = B = X1 * Y1 MMOV32 MRO, *MARO[2]++ ; In parallel MRO = X2, MARO += 2 MMOV32 MR1, *MARO[2]++ ; In parallel MRO = X2, MARO += 2 MMOV32 MR1, *MARO[2]++ ; In parallel MRO = X2, MARO += 2 MMACF32 MR3, MR2, MR2, MR0, MRI ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR1, *MARO[2]++ ; In parallel MR0 = X3 MMACF32 MR3, MR2, MR2, MR0, MRI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MRO, *MARO ; In parallel MR0 = X4 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MARO ; In parallel MR0 = X4 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = Y4 MMOV32 MR1, *MARI ; MR2 = Y4 MMOV32 MR3, MR3, MR2 ; In parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; Im parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR3 ; Store the result MSTOP ; end of task</pre>			MMOVI16	MAR0,	#_X	;	MAR0 points to X array
<pre>MNOP ; Delay For MARO, MARI load MNOP ; Delay For MARO, MARI load ; < MARO valid MMOV32 MRO, *MARO[2]++ ; MRO = XO, MARO += 2 ; < MARO valid MMOV32 MR1, *MARI[2]++ ; MR1 = YO, MARI += 2 MMPYF32 MR2, MRO, MR1 ; MR2 = A = XO * YO MMOV32 MRO, *MARO[2]++ ; In parallel MRO = X1, MARO += 2 MMPYF32 MR3, MRO, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR1, *MARI[2]++ ; In parallel MRO = X2, MARO += 2 MMPYF32 MR3, MRO, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR1, *MARI[2]++ ; In parallel MRO = X2, MARO += 2 MMOV32 MR1, *MARI[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR1, *MARI[2]++ ; In parallel MRO = X3 MMOV32 MR1, *MARI[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MARI ; MR3 = (A + B) + C, MR2 = D = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR3 ; Store the result MTOV32 @_Result, MR3 ; Store the result ; end of task</pre>			MMOVI16	MARI,	#_Y	;	MARI points to Y array
<pre>MNOP</pre>			MNOP			΄.	Delay for MARO, MARI 10ad
<pre>MMOV32 MR0, *MAR0[2]++ ; MR0 = X0, MAR0 += 2 ; < MAR1 valid MMOV32 MR1, *MAR1[2]++ ; MR1 = Y0, MAR1 += 2 MMPYF32 MR2, MR0, MR1 ; MR2 = A = X0 * Y0 [MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X1, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 [MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 [MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 [MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 [MMOV32 MR1, *MAR1 ; MR1 = Y4 MMDV32 MR1, *MAR1 ; MR2 = E = X4 * Y4 [MADDF32 MR3, MR3, MR2 ; In parallel MR3 = (A + B + C) + D MADF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task</pre>			MINOP			;	<pre>MARO valid</pre>
<pre>MAGUS2 MR0, MR0, MR1 ; , < MR1 valid</pre>			MMOV32	MRO	*MAR0[2]++	;	MR0 = X0 MAR0 += 2
<pre>MMOV32 MR1, *MAR1[2]++ ; MR1 = Y0, MAR1 += 2 MMPYF32 MR2, MR0, MR1 ; MR2 = A = X0 * Y0 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X1, MAR0 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR1, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MAR1 ; MR2 = E = X4 * Y4 MMOV32 MR1, *MAR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task</pre>			1110102	,	11110[2]	;	< MAR1 valid
<pre>MMPYF32 MR2, MR0, MR1 ; MR2 = A = X0 * Y0 MM0V32 MR0, *MAR0[2]++ ; In parallel MR0 = X1, MAR0 += 2 MM0V32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 MM0V32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MM0V32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MM0V32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MM0V32 @_Result, MR3 ; Store the result ; end of task</pre>			MMOV32	MR1,	*MAR1[2]++	;	MR1 = Y0, MAR1 += 2
<pre> MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X1, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result ; end of task</pre>			MMPYF32	MR2,	MRO, MR1	;	MR2 = A = X0 * Y0
<pre>MMOV32 MR1, *MAR1[2]++ ; MR1 = Y1, MAR1 += 2 MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result ; end of task </pre>			MMOV32	MR0,	*MAR0[2]++	;	In parallel MRO = X1, MARO += 2
<pre>MMPYF32 MR3, MR0, MR1 ; MR3 = B = X1 * Y1 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMOV32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR2, MR2, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result ; end of task</pre>			MMOV32	MR1,	*MAR1[2]++	;	MR1 = Y1, MAR1 += 2
<pre> MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X2, MAR0 += 2 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result</pre>			MMPYF32	MR3,	MRO, MR1	;	MR3 = B = X1 * Y1
<pre>MMOV32 MR1, *MAR1[2]++ ; MR1 = Y2, MAR2 += 2 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MMPYF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result ; end of task</pre>			MMOV32	MR0,	*MAR0[2]++	;	In parallel MRO = X2, MARO += 2
<pre>MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2 MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MMPYF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task</pre>			MMOV32	MR1,	*MAR1[2]++	;	MR1 = Y2, MAR2 += 2
<pre> MMOV32 MR0, *MAR0[2]++ ; In parallel MR0 = X3 MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task</pre>			MMACF32	MR3,	MR2, MR2, MR0, MR1	;	MR3 = A + B, MR2 = C = X2 * Y2
<pre>MMOV32 MR1, *MAR1[2]++ ; MR1 = Y3 MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR2 = E = X4 * Y4 MADDF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; end of task</pre>			MMOV32	MR0,	*MAR0[2]++	;	In parallel MRO = X3
MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3 MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR2 = E = X4 * Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; Store the result MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task			MMOV32	MR1,	*MAR1[2]++	;	MR1 = Y3
<pre> MMOV32 MR0, *MAR0 ; In parallel MR0 = X4 MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task</pre>			MMACF32	MR3,	MR2, MR2, MR0, MR1	;	MR3 = (A + B) + C, MR2 = D = X3 * Y3
MMOV32 MR1, *MAR1 ; MR1 = Y4 MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task			MMOV32	MR0,	*MAR0	;	In parallel MRO = X4
MMPYF32 MR2, MR0, MR1 ; MR2 = E = X4 * Y4 MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task			MMOV32	MR1,	*MAR1	;	MR1 = Y4
<pre> MADDF32 MR3, MR3, MR2 ; in parallel MR3 = (A + B + C) + D MADDF32 MR3, MR3, MR2 ; MR3 = (A + B + C + D) + E MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task</pre>			MMPYF32	MR2,	MRO, MR1	;	MR2 = E = X4 * Y4
MADDF32MR3, MR3, MR2; MR3 = (A + B + C + D) + EMMOV32@_Result, MR3; Store the resultMSTOP; end of task			MADDF32	MR3,	MR3, MR2	;	in parallel MR3 = $(A + B + C) + D$
MMOV32 @_Result, MR3 ; Store the result MSTOP ; end of task			MADDF32	MR3,	MR3, MR2	;	MR3 = (A + B + C + D) + E
MSTOP ; end of task			MMOV32	@_Res	sult, MR3	;	Store the result
			MSTOP			;	end of task

See also

MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32



MMPYF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 32-bit Floating-Point Multiply with Parallel Move

Operands						
	MRd	CLA floating MRd cannot	-point destination register fo be the same register as MR	r the MMPYF32 (MR Ra	0 to MR3)	
	MRe	CLA floating	-point source register for the	MMPYF32 (MR0 to	MR3)	
	MRf	CLA floating	-point source register for the	MMPYF32 (MR0 to	MR3)	
	MRa	CLA floating MRa cannot	-point destination register fo be the same register as MR	r the MMOV32 (MR0 Rd	to MR3)	
	mem32	32-bit memo source of the	ry location accessed using on MMOV32.	direct or indirect addr	essing. This will be the	
Opcode	LSW: mmmm mm MSW: 0000 ff	mm mmmm mmmm ee ddaa addr				
Description	Multiply the of MRd = MRe * MRa = [mem32	contents of two f	loating-point registers	and load anothe	r.	
Restrictions	The destinat MRa cannot	ion register for t be the same reg	he MMPYF32 and the gister as MRd.	MMOV32 must b	be unique. That is,	
Flags	This instructi	on modifies the	following flags in the N	/ISTF register:.		
	Flag	TF ZF	NF	LUF	LVF	
	Modified	No Ye	s Yes	Yes	Yes	
	The MSTF re • LUF = 1 i • LVF = 1 i The MMOV3 NF = MRa(31) ZF = 0;	egister flags are f MMPYF32 ger f MMPYF32 ger 2 Instruction wil	modified as follows: nerates an underflow conterates an overflow conterates an overflow conterates the NF and ZF fla	ondition. ndition. Igs as follows:		
	11(MRa(30:23	$() == 0) \{ ZF' =$	$\perp i \text{ NF} = 0i \}$			
Pipeline	Both MMPY	-32 and MMOV	32 complete in a single	e cycle.		
Example 1	; Given M1, ; Calculate ; _ClalTask1: _MMOV32 MMOV32 MMPYF32 MMOV32 MADDF32 MMOV32 MSTOD	X1 and B1 are 3 Y1 = M1*X1+B1 MR0, @M1 MR1, @X1 MR1, MR1, MI MR0, @B1 MR1, MR1, MI @Y1, MR1	32-bit floating point ; Load MR0 with ; Load MR1 with R0 ; Multiply M1*X1 ; and in paralle R0 ; Add M*X1 to B1 ; Store the resu ; end of task	M1 X1 l load MR0 with and store in M lt	B1 R1	

Instruction Set

www.ti.com	
Example 2	<pre>; Given A, B and C are 32-bit floating-point numbers ; Calculate Y2 = (A * B) ; Y3 = (A * B) * C ; ClalTask2:</pre>
	MMOV32 MR0, @A ; Load MR0 with A MMOV32 MR1, @B ; Load MR1 with B MMPYF32 MR1, MR1, MR0 ; Multiply A*B MMOV32 MR0, @C ; and in parallel load MR0 with C MMPYF32 MR1, MR1, MR0 ; Multiply (A*B) by C MMOV32 @Y2, MR1 ; and in parallel store A*B MMOV32 @Y3, MR1 ; Store the result MSTOP ; end of task
See also	MMPYF32 MRd, MRe, MRf MMOV32 mem32, MRa MMACF32 MR3, MR2, MRd, MRe, MRf MMOV32 MRa, mem32



MMPYF32 MRd, MRe, MRf ||MMOV32 mem32, MRa 32-bit Floating-Point Multiply with Parallel Move

Operands						
	MRd	CLA floating-	point destination register for	r the MMPYF32 (MR	0 to MR3)	
	MRe	CLA floating-	point source register for the	MMPYF32 (MR0 to	MR3)	
	MRf	CLA floating-	point source register for the	MMPYF32 (MR0 to	MR3)	
	mem32	32-bit memor destination of	ry location accessed using of the MMOV32.	direct or indirect addre	essing. This will be the	
	MRa	CLA floating-	point source register for the	MMOV32 (MR0 to M	1R3)	
Opcode	LSW: mmmm mmm MSW: 0100 ffe	um mmmm mmmm ee ddaa addr				
Description	Multiply the contents of two floating-point registers and move from memory to register. MRd = MRe * MRf; [mem32] = MRa;					
Flags	This instruction	on modifies the f	following flags in the M	ISTF register:		
	Flag 1	F ZF	NF	LUF	LVF	
	Modified N	No No	No	Yes	Yes	
Pipeline	 LUF = 1 if LVF = 1 if MMPYF32 an 	MMPYF32 gen MMPYF32 gen d MMOV32 bot	erates an underflow c erates an overflow co h complete in a single	ondition. ndition. • cycle.		
			, ,	,		
Example	; Given A, B ; Calculate Y ; Y ; _ClalTask2: MMOV32 MMOV32 MMPYF32 MMOV32 MMPYF32 MMOV32 MMOV32 MMOV32 MSTOP	and C are 32-b (2 = (A * B) (3 = (A * B) * MR0, @A MR1, @B MR1, MR1, MR MR0, @C MR1, MR1, MR @Y2, MR1 @Y3, MR1	<pre>pit floating-point nu C ; Load MR0 with A ; Load MR1 with B ; and in parallel ; Multiply A*B ; and in parallel ; and in parallel ; Store the resul ; end of task</pre>	load MR0 with C by C store A*B t	2	
See also	MMPYF32 MI MMACF32 M	Rd, MRe, MRf R3, MR2, MRd,	MMOV32 MRa, mem MRe, MRf MMOV32	132 2 MRa, mem32		

MMPYF32 MRa, MRb, MRc ||MSUBF32 MRd, MRe, MRf 32-bit Floating-Point Multiply with Parallel Subtract

Operands					
	MRa	CLA floating-pe MRa cannot be	oint destination register for the same register as MR	MMPYF32 (MR0 to	MR3)
	MRb	CLA floating-p	oint source register for MM	IPYF32 (MR0 to MR	3)
	MRc	CLA floating-p	oint source register for MM	IPYF32 (MR0 to MR	3)
	MRd	CLA floating-pe MRd cannot be	oint destination register for e the same register as MR	MSUBF32 (MR0 to a	MR3)
	MRe	CLA floating-p	oint source register for MS	UBF32 (MR0 to MR3	3)
	MRf	CLA floating-p	oint source register for MS	UBF32 (MR0 to MR3	3)
Dpcode	LSW: 0000 ffe MSW: 0111 10	ee ddcc bbaa 10 0100 0000			
Description	Multiply the c registers. MRa = MRb * MRd = MRc - M	ontents of two flo MRc; MRf;	ating-point registers v	with parallel subt	raction of two
Restrictions	The destination MRa cannot b	on register for the	e MMPYF32 and the I ster as MRd.	MSUBF32 must	be unique. That is,
Flags	This instruction	on modifies the fo	ollowing flags in the M	ISTF register:.	
-	Flag	TF ZF	NF	LUF	LVF
	Modified I	No No	No	Yes	Yes
	The MSTF re	gister flags are m	nodified as follows:		
	 LUF = 1 if LVF = 1 if 	MMPYF32 or M MMPYF32 or M	SUBF32 generates a SUBF32 generates ar	n underflow cond n overflow condit	lition. ion.
Pipeline	MMPYF32 ar	nd MSUBF32 bot	h complete in a single	e cycle.	
Example	; Given A, B ; Calculate ; ; ;	and C are 32-bi f2 = (A * B) f3 = (A - B)	t floating-point num	nbers	
	_Cla1Task2: MMOV32 MMOV32 MMPYF32 MSUBF32 MMOV32 MMOV32 MSTOP	MR0, @A MR1, @B MR2, MR0, MR1 MR3, MR0, MR1 @Y2, MR2 @Y3, MR3	; Load MR0 with A ; Load MR1 with B ; Multiply (A*B) ; and in parallel ; Store A*B ; Store A-B ; end of task	Sub (A-B)	
See also	MSUBF32 MI MSUBF32 MI MSUBF32 MI	Ra, MRb, MRc Rd, MRe, MRf Rd, MRe, MRf	MMOV32 MRa, mem MMOV32 mem32, MF	32 Ra	

MNEGF32 MRa, MRb{, CNDF} Conditional Negation

Operands

e por an ao					
	MRa	CLA floatir	ng-point destination register (MR0 to M	२३)	
	MRb	CLA floatir	ng-point source register (MR0 to MR3)		
	CNDF	condition t	ested		
Opcode	LSW: 0000 MSW: 0111	0000 cndf bbaa 1010 1000 0000			
Description	if (CNDF == true) {MRa = - MRb; } else {MRa = MRb; }				
	CNDF is o	ne of the followir	ng conditions:		
	Encode ⁽⁵⁾	CNDF	Description	MSTF 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 OR 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 ⁽⁶⁾	Unconditional with flag modification	None	
	 ⁽⁵⁾ Values n ⁽⁶⁾ This is th be modified 	ot shown are reserve e default operation if ied when a condition	ed. f no CNDF field is specified. This condi al operation is executed. All other cond	tion will allow the ZF, and N itions will not modify these	F flags to flags.
Flags	This instru	ction modifies th	e following flags in the MSTF re	egister:	
	Flag	TE 7			

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

Pipeline

Example 1

This is a single-cycle instruction.

; Show the basic operation of $\ensuremath{\mathsf{MNEGF32}}$

MRO,	#5.0	;	$MR0 = 5.0 (0 \times 40 \land 0000)$
MR1,	#4.0	;	$MR1 = 4.0 (0 \times 40800000)$
MR2,	#-1.5	;	$MR2 = -1.5 (0 \times BFC00000)$
MR3,	MR1, MR2	;	MR3 = -6.0
MR0,	MRO, MR1	;	MR0 = 20.0
MR1,	#0.0		
MR3,	MR1	;	NF = 1
MR3,	MR3, LT	;	if NF = 1, MR3 = 6.0
MR0,	MR1	;	NF = 0
MR0,	MR0, GEQ	;	if NF = 0, MR0 = -20.0
	MR0, MR1, MR2, MR3, MR0, MR1, MR3, MR3, MR0,	MR0, #5.0 MR1, #4.0 MR2, #-1.5 MR3, MR1, MR2 MR0, MR0, MR1 MR1, #0.0 MR3, MR1 MR3, MR1 MR3, MR1 MR0, MR1 MR0, MR0, GEQ	MRO, #5.0 ; MR1, #4.0 ; MR2, #-1.5 ; MR3, MR1, MR2 ; MR0, MR0, MR1 ; MR1, #0.0 MR3, MR1 ; MR3, MR3, LT ; MR0, MR1 ; MR0, MR0, GEQ ;

;



Example 2	<pre>; Calculate N ; Ye = Estima ; Ye = Ye*(2. ; Ye = Ye*(2. ; ClalTask1:</pre>	um/Den using a No te(1/X) 0 - Ye*X) 0 - Ye*X)	ewton-Raphson algorithum for 1/Den
	_Claffaski MMOV32 MEINVF32 MSUBF32 MSUBF32 MMPYF32 MMPYF32 MMOV32 MSUBF32 MMPYF32 MMEGF32 MMPYF32 MMOV32 MSTOP	MR1, @_Den MR2, MR1 MR3, MR2, MR1 MR3, #2.0, MR3 MR2, MR2, MR3 MR3, MR2, MR1 MR0, @_Num MR3, #2.0, MR3 MR2, MR2, MR3 MR1, @_Den MR0, MR0, EQ MR0, MR2, MR0 @_Dest, MR0	<pre>; MR1 = Den ; MR2 = Ye = Estimate(1/Den) ; MR3 = Ye*Den ; MR3 = 2.0 - Ye*Den ; MR3 = 2.0 - Ye*Den ; MR2 = Ye = Ye*(2.0 - Ye*Den) ; MR3 = Ye*Den ; MR0 = Num ; MR3 = 2.0 - Ye*Den ; MR2 = Ye = Ye*(2.0 - Ye*Den) ; Reload Den To Set Sign ; if(Den == 0.0) Change Sign Of Num ; MR0 = Y = Ye*Num ; Store result ; end of task</pre>

See also

MABSF32 MRa, MRb



Instruction S	et
---------------	----

MNOP	No Operation								
Operands									
	none	This instruction	does not have any opera	nds					
Opcode	LSW: 0000 00 MSW: 0111 11	LSW: 0000 0000 0000 MSW: 0111 1111 1010 0000							
Description	Do nothing. T instructions a	Do nothing. This instruction is used to fill required pipeline delay slots when other instructions are not available to fill the slots.							
Flags	This instruction does not modify flags in the MSTF register.								
	Flag	TF ZF	NF	LUF	LVF				
	Modified	No No	No	No	No				
	_ClalTask1: MMOVI16 MUI16TOF MNOP	_ClalTask1: MMOVI16 MAR1,#_X ; Start address MUI16TOF32 MR0, @_len ; Length of the array MNOP : delay for MAR1 load							
	MNOP MMOV32	MR1. *MAR1[2]+	; delay for MAR + ; MR1 = X0	1 load					
	LOOP MMOV32 MMAXF32 MADDF32 MCMPF32 MNOP MNOP	MR2, *MAR1[2]+ MR1, MR2 MR0, MR0, #-1. MR0 #0.0	<pre>+ ; MR2 = next el. ; MR1 = MAX(MR1 0 ; Decrement the ; Set/clear fla ; Too late to a ; Too late to a</pre>	ement , MR2) counter gs for MBCNDD ffect MBCNDD ffect MBCNDD					

See also

MOR32 MRa, MRb, MRc Bitwise OR

	MRa	CL	A floating-point de	stination register (M	R0 to MR3)	
	MRb	CL	A floating-point so	urce register (MR0 t	o MR3)	
	MRc	CL	A floating-point so	urce register (MR0 t	o MR3)	
Opcode	LSW: 000 MSW: 011	0 0000 00cc 1 1100 1000	bbaa 0000			
Description	Bitwise C	OR of MRb v 0) = MARb(3	vith MRc. 1:0) OR MRc(31	:0);		
Flags	This inst	ruction modi	fies the followir	ng flags in the M	STF register:	
	Flag	TF	ZF	NF	LUF	LVF
	Modified	No	Yes	Yes	No	No
Pipeline	This is a	single-cycle	instruction.			
Pipeline	This is a	single-cycle	instruction.			
Example	MMOVIZ MMOVXI	MR0, #0x5 MR0, #0xA	555 ; MRO = Ox AAA	5555AAAA		
	MMOVIZ MMOVXI	MR1, #0x5 MR1, #0xF	432 ; MR1 = 0x EDC	5432FEDC		
			; 0101 OR ; 0101 OR ; 0101 OR ; 0101 OR ; 1010 OR ; 1010 OR ; 1010 OR ; 1010 OR	0101 = 0101 (5) 0100 = 0101 (5) 0011 = 0111 (7) 0110 = 0111 (7) 1111 = 1111 (F) 1110 = 1110 (E) 1101 = 1111 (F) 1100 = 1110 (E)		
	MOR32 MR	2, MR1, MR0	; MR3 = 0x	5555FEFE		
See also	MAND32 MXOR32	2 MRa, MRb 2 MRa, MRb	, MRc , MRc			



MRCNDD {CNDF} Return Conditional Delayed

CNDF

Operands

Description

optional condition.

Opcode	LSW:	0000	0000	0000	0000	
	MSW:	0111	1001	1010	cndf	

If the specified condition is true, then the RPC field of MSTF is loaded into MPC and fetching continues from that location. Otherwise program fetches will continue without the return.

Please refer to the pipeline section for important information regarding this instruction. if (CNDF == TRUE) MPC = RPC;

CNDF is one of the following conditions:

	0		
Encode (7)	CNDF	Description	MSTF 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 OR 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 ⁽⁸⁾	Unconditional with flag modification	None

⁽⁷⁾ Values not shown are reserved.

⁽⁸⁾ This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

Instruction Set The MRCNDD instruction by itself is a single-cycle instruction. As shown in Table 31, for each return 6 instruction slots are executed; three before the return instruction (d5-d7) and three after the return instruction (d8-d10). The total number of cycles for a return taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a return can, therefore, range from 1 to 7 cycles. The number of cycles for a return taken may not be the same as for a return not taken. Referring to the following code fragment and the pipeline diagrams in Table 31 and Table 32, the instructions before and after MRCNDD have the following properties: ; <Instruction 1> ; Il Last instruction that can affect flags for ; the MCCNDD operation <Instruction 2> <Instruction 3> ; I2 Cannot be stop, branch, call or return ; I3 Cannot be stop, branch, call or return <Instruction 4> ; I4 Cannot be stop, branch, call or return MCCNDD _func, NEQ ; Call to func if not eqal to zero ; Three instructions after MCCNDD are always ; executed whether the call is taken or not <Instruction 5> ; I5 Cannot be stop, branch, call or return <Instruction 6> ; I6 Cannot be stop, branch, call or return <Instruction 7> ; I7 Cannot be stop, branch, call or return <Instruction 8> ; I8 The address of this instruction is saved ; in the RPC field of the MSTF register. ; Upon return this value is loaded into MPC ; and fetching continues from this point. <Instruction 9> ; I9 <Instruction 10> ; I10 func: <Destination 1> ; dl Can be any instruction <Destination 2> ; d2 <Destination 3> ; d3 <Destination 4> ; d4 Last instruction that can affect flags for ; the MRCNDD operation <Destination 5> ; d5 Cannot be stop, branch, call or return
<Destination 6> ; d6 Cannot be stop, branch, call or return <Destination 7> ; d7 Cannot be stop, branch, call or return ; Return to <Instruction 8> if not equal to zero MRCNDD, NEO ; Three instructions after MRCNDD are always constinuation 8> ; executed whether the return is taken or not constinuation 8> ; d8 Cannot be stop, branch, call or return constinuation 9> ; d9 Cannot be stop, branch, call or return <Destination 10> ; d10 Cannot be stop, branch, call or return <Destination 11> ; d11 <Destination 12> ; d11

MSTOP

. . . .

• d4

- d4 is the last instruction that can effect the CNDF flags for the MRCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to return or not when MRCNDD is in the D2 phase.
- There are no restrictions on the type of instruction for d4.
- d5, d6 and d7
 - The three instructions proceeding MRCNDD can change MSTF flags but will have no effect on whether the MRCNDD instruction makes the return or not. This is because the flag modification will occur after the D2 phase of the MRCNDD instruction.
 - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.



• d8, d9 and d10

- The three instructions following MRCNDD are always executed irrespective of whether the return is taken or not.
- These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

Instruction	F1	F2	D1	D2	R1	R2	E	W
d4	d4	d3	d2	d1	17	16	15	
d5	d5	d4	d3	d2	d1	17	16	
d6	d6	d5	d4	d3	d2	d1	i7	
d7	d7	d6	d5	d4	d3	d2	d1	
MRCNDD	MRCNDD	d7	d6	d5	d4	d3	d2	
d8	d8	MRCNDD	d7	d6	d5	d4	d3	
d9	d9	d8	MRCNDD	d7	d6	d5	d4	
d10	d10	d9	d8	MRCNDD	d7	d6	d5	
d11	d11	d10	d9	d8	-	d7	d6	
d12	d12	d11	d10	d9	d8	-	d7	
etc		d12	d11	d10	d9	d8	-	
			d12	d11	d10	d9	d8	
				d12	d11	d10	d9	
					d12	d11	d10	
						d12	d11	
							d12	

Table 32. Pipeline Activity For MRCNDD, Return Taken

Instruction	F1	F2	D1	D2	R1	R2	Е	w
d4	d4	d3	d2	d1	17	16	15	
d5	d5	d4	d3	d2	d1	17	16	
d6	d6	d5	d4	d3	d2	d1	i7	
d7	d7	d6	d5	d4	d3	d2	d1	
MRCNDD	MRCNDD	d7	d6	d5	d4	d3	d2	
d8	d8	MRCNDD	d7	d6	d5	d4	d3	
d9	d9	d8	MRCNDD	d7	d6	d5	d4	
d10	d10	d9	d8	MRCNDD	d7	d6	d5	
18	18	d10	d9	d8	-	d7	d6	
19	19	18	d10	d9	d8	-	d7	
110	I10	19	18	d10	d9	d8	-	
etc		l10	19	18	d10	d9	d8	
			l10	19	18	d10	d9	
				I10	19	18	d10	
					I10	19	18	
						l10	19	
							l10	

Example

See also

MBCNDD #16BitDest, CNDF MCCNDD 16BitDest, CNDF MMOV32 mem32, MSTF MMOV32 MSTF, mem32

;



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

Operands										
	FLAG	8	bit mask in	dicating whi	ch floating-p	oint status f	ags to chang	je.		
	VALUE	8	8 bit mask in	dicating the	flag value; 0	or 1.				
Opcode	LSW: FFFF MSW: 0111	LSW: FFFF FFFF VVVV VVVV MSW: 0111 1001 1100 0000								
Description	The MSE the MSTF changed. flags will r	TFLG inst register. That is, if not be mo	ruction is The FLA0 a FLAG I dified. Th	used to s G field is a bit is set to e bit map	et or clear an 11-bit v o 1 it indic oing of the	selected alue that ates that FLAG fie	floating-po indicates v flag will be eld is show	oint status which flags changed; n below:	flags in will be all other	
	reserved	RNDF32	TF	reserved	reserved	ZF	NF	LUF	LVF	
	8	7	6	5	4	3	2	1	0	
Flags	The VALU	JE field in uction mo	dicates th difies the	e value th	e flag sho flags in the	ould be se e MSTF r	et to; 0 or 1 egister:			
	Flag	TF	ZF		NF		LUF	LVF		
	Modified	Yes	Ye	S	Yes		Yes	Yes		
	Any flag o modified v	an be mo with this ir	dified by struction.	this instru	ction. The	MEALLC	W and RF	PC fields ca	annot be	
Pipeline	This is a s	single-cyc	le instruct	ion.						
Example	To make i MSTFLG MSETFLG	it easier a operation	nd legible as showr TF=0, NF	e, the asse n below: F=1; FLAG	embler acc	cepts a Fl	_AG=VALU = 00xxx1x	JE syntax f	or the	
See also	MMOV32 MMOV32	mem32, MSTF, m	MSTF em32							



Instruction Set						www.ti.com		
MSTOP	Stop Task							
Operands								
	none	Т	his instruction does	not have any operar	nds			
Opcode	LSW: 0000 MSW: 0111	0000 000 1111 100	0 0000 0 0000					
Description	The MSTO placing MS for debugg of the pipe flagged in t	P instruc TOP in t ing and p line, the the PIE v	tion must be pla unused memory preventing run a MIRUN flag for rector table.	aced to indicate t locations within way CLA code. \ the task is cleare	he end of each the CLA progra When MSTOP of and the asso	task. In addition, im RAM can be useful enters the D2 phase ciated interrupt is		
	There are t MPC reach	three spe nes the N	ecial cases that ISTOP instruction	can occur when a	single-stepping	a task such that the		
	 If you are single-stepping or halted in "task A" and "task B" comes in before the MPC reaches the MSTOP, then "task B" will start if you continue to step through the MSTOP instruction. Basically if "task B" is pending before the MPC reaches MSTOP in "task A" then there is no issue in "task B" starting and no special action is required 							
	2. In this of the MS in the M the MS in. To re this is o	case you TOP with IIFR regi TOP inst eliably st lone, you	have single-ste no tasks pendi ster but it may o ruction of "task art "task B" perf can start single	pped or halted in ng. If "task B" co or may not start if A". It depends or orm a soft reset a e-stepping "task I	"task A" and the mes in at this p f you continue to exactly when and reconfigure 3".	he MPC has reached ooint, it will be flagged o single-step through the new task comes the MIER bits. Once		
	3. Case 2 in (for e single-s tasks p state. C	can be h example of stepped c ending. E Once this	nandled slightly using the IACK or halted in "task Before forcing "t is done you car	differently if there instruction to star (A" and the MPC ask B", run free t n force "task B" a	e is control over rt the task). In the C has reached to o force the CLA and continue de	when "task B" comes his case you have he MSTOP with no out of the debug bugging.		
Restrictions	The MSTO MCCNDD	P instructor or MRCN	tion cannot be	placed 3 instructi	ons before or a	fter a MBCNDD,		
Flags	This instrue	ction doe	s not modify fla	gs in the MSTF r	egister.			
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	No	No	No	No		
Pipeline	This is a si instruction. MCCNDD	ngle-cycl The MS or MRCN	e instruction. Ta TOP instruction	able 33 shows the cannot be place	e pipeline beha d with 3 instruc	vior of the MSTOP tions of a MBCNDD,		

TEXAS INSTRUMENTS

www.t	i.com
-------	-------

Instruction Set

Instruction	F1	F2	D1	D2	R1	R2	Е	w
11	l1							
12	12	l1						
3	13	12	11					
MSTOP	MSTOP	13	12	11				
4	14	MSTOP	13	12	l1			
5	15	14	MSTOP	13	12	11		
6	16	15	14	MSTOP	13	12	l1	
New Task Arbitrated and Piroitized	-	-	-	-	-	13	12	
New Task Arbitrated and Piroitized	-	-	-	-	-	-	13	
1	l1	-	-	-	-	-	-	
2	12	11	-	-	-	-	-	
3	13	12	11	-	-	-	-	
4	14	13	12	11	-	-	-	
5	15	14	13	12	l1	-	-	
6	16	15	14	13	12	11	-	
7	17	16	15	14	13	12	l1	
etc								

Example

; Given A = ; B = ; C = ;	(int32)1 (int32)2 (int32)-7
; Calculate	Y2 = A - B - C
_ClalTask3:	MR0, @_A ; MR0 = 1 (0x00000001)
MMOV32	MR1, @_B ; MR1 = 2 (0x00000002)
MMOV32	MR2, @_C ; MR2 = -7 (0xFFFFFF9)
MSUB32	MR3, MR0, MR1 ; A + B
MSUB32	MR3, MR3, MR2 ; A + B + C = 6 (0x000006)
MMOV32	@_y2, MR3 ; Store y2
MSTOP	; End of task

See also

MDEBUGSTOP



Instruction Set

MSUB32 MRa, MRb, MRc 32-bit Integer Subtraction

operanae									
	MRa CLA floating-point destination register (MR0 to MR3)								
	MRb CLA floating-point destination register (MR0 to MR3)								
	MRc	CLA	floating-point de	stination register (MI	R0 to MR3)				
Opcode	LSW: 0000 MSW: 0111	0000 00cc b 1100 1110 0	baa 000						
Description	32-bit inte	ger addition	of MRb and N 0) - MRc(31:	//Rc. 0);					
Flags	This instru	ction modifie	es the followir	ng flags in the M	STF register:				
	Flag	TF	ZF	NF	LUF	LVF	_		
	Modified	No	Yes	Yes	No	No	_		
	The MSTF NF = MRa(3 ZF = 0; if(MRa(31)	register flag 31); :0) == 0) {	gs are modifie ZF = 1; }	ed as follows:					
Pipeline	This is a s	ingle-cycle ir	nstruction.						
Example	; Given A ; B ; C ; Calculate ; _ClalTask: MMOV32 MMOV32 MMOV32 MSUB32 MSUB32 MMOV32 MSUB32	<pre>= (int32)1 = (int32)2 = (int32)-7 Y2 = A - B 3: MR0, @_A MR1, @_B MR2, @_C MR3, MR0, MR3, MR3, @_Y2, MR3</pre>	- C ; M ; M ; M MR1 ; A MR2 ; A ; S ; E	R0 = 1 (0x00000 R1 = 2 (0x00000 R2 = -7 (0xFFFF + B + B + C = 6 (0 tore y2 nd of task	001) 002) FFF9) x0000006)				
See also	MADD32 MAND32 MASR32 MLSL32 N MLSR32 N MOR32 M MXOR32	MRa, MRb, M MRa, MRb, M MRa, #SHIF MRa, #SHIFT MRa, #SHIFT Ra, MRb, M MRa, MRb, M	MRc MRc T Rc MRc						

MSUBF32 MRa, MRb, MRc 32-bit Floating-Point Subtraction

Operands							
	MRa	С	LA floating-point de	estination register (M	R0 to R1)		
	MRb	С	LA floating-point so	urce register (MR0 t	o R1)		
	MRc	С	LA floating-point so	urce register (MR0 t	o R1)		
Opcode	LSW: 0000 MSW: 0111	0000 00cd 1100 0100	c bbaa 0 0000				
Description	Subtract the MRa = MRb	e content	ts of two floating	g-point registers			
Flags	This instruc	ction mod	lifies the followi	ng flags in the M	ISTF register:		
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	Yes	Yes	
Pipeline	This is a si	ngle-cycl	e instruction.				
Pipeline Example	<pre>This is a sin ; Given A, ; Calculat ; _ClalTask5 MMOV32 MMOV32 MADDF32 MMOV32 MSUBF32 WSUBF32</pre>	B and C e Y2 = A : MR0, 0 MR1, 0 MR1, 0 MR0, 1	e instruction. are 32-bit flo + B - C @_A ; Load @_B ; Load WR1, MR0 ; Add @_C ; ar WR0, MR1 ; Subt	ating-point num A MRO with A A MR1 with B A + B ad in parallel 1 rract C from (A	nbers Load C + B)		
See also	MMOV32 MSTOP MSUBF32 MSUBF32 MSUBF32 MMPYF32	[@] ¥, ™ MRa, #10 MRd, MF MRd, MF MRa, MF	MR0 ; (A+F ; end 6FHi, MRb Re, MRf MMO Re, MRf MMO Rb, MRc MSU	³⁾ - C of task V32 MRa, mem V32 mem32, MF IBF32 MRd, MR	32 Ra e, MRf		



MSUBF32 MRa, #16FHi, MRb 32-bit Floating Point Subtraction

\sim	-		-	_	-
U	De	Гd	n	u	

Operands						
	MRa	CLA float	ing-point dest	ination register (M	R0 to R1)	
	#16FHi	A 16-bit in floating-p	mmediate val oint value. Th	ue that represents e low 16-bits of the	the upper 16-bits of e mantissa are assur	an IEEE 32-bit med to be all 0.
	MRb	CLA float	ing-point sour	ce register (MR0 t	o R1)	
Opcode	LSW: IIII I MSW: 0111 10	III IIII IIII)00 0000 baaa				
Description	Subtract MR the result of	b from the float the addition ir	ating-point n MRa.	value represer	nted by the imme	diate operand. Store
	#16FHi is a floating-poin most useful Some exam (0xBFC0000 That is, the MRa = #16FH:	16-bit immedia t value. The lo for representin ples are 2.0 (0 0). The asser value -1.5 can i:0 - MRb;	ate value th ow 16-bits on ng constan 0x4000000 nbler will a be represe	hat represents of the mantissa is where the lo 0), 4.0 (0x4080 ccept either a l ented as #-1.5	the upper 16-bits a are assumed to west 16-bits of th 00000), 0.5 (0x3F nex or float as th or #0xBFC0.	s of an IEEE 32-bit be all 0. #16FHi is he mantissa are 0. -000000), and -1.5 e immediate value.
Flags	This instruct	ion modifies th	ne following	g flags in the M	STF register:	
	Flag	TF	ZF	NF	LUF	LVF
	Modified	No	No	No	Yes	Yes
Pipeline	 LVF = 1 LVF = 1 This is a single 	if MSUBF32 g gle-cycle instr	jenerates a uction.	n overflow con	dition.	
Example	; Y = sqrt() ; Ye = Estin ; Ye = Ye*() ; Ye = Ye*() ; Y = X*Ye ; _ClalTask3: _MMOV32 _MEISQRTF3 _MMOV32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32 _MMPYF32	<pre>X) nate(1/sqrt(X 1.5 - Ye*Ye*X 1.5 - Ye*Ye*X 32 MR1, MR0, MR1, @_x, MR3, MR0, MR2, MR1, MR2</pre>)); *0.5) *0.5) EQ ; #0.5 ; MR2 ;	MR0 = X MR1 = Ye = Es if(X == 0.0) MR3 = X*0.5 MR2 = Ye*X*0. MR2 = Ye*Ye*2 MR1 = Ye = Ye MR2 = Ye*X*0. MR2 = Ye*Ye*2 MR2 = Ye*Ye*2 MR2 = 1.5 - 3 MR1 = Ye = Ye* MR0 = Y = Ye* Store Y = squ end of task	stimate(1/sqrt(X Ye = 0.0 5 (*0.5 Ye*Ye*X*0.5 **(1.5 - Ye*Ye*X 5 (*0.5 Ye*Ye*X*0.5 **(1.5 - Ye*Ye*X Ya Ya Ya Ya Ya Ya Ya Ya Ya Ya Ya Ya Ya	<pre>()) (*0.5) (*0.5)</pre>
See also	MSUBF32 M MSUBF32 M MSUBF32 M MMPYF32 M	IRa, MRb, MF IRd, MRe, MF IRd, MRe, MF IRa, MRb, MF	Rc Rf MMOV Rf MMOV Rc MSUE	32 MRa, mem 32 mem32, MF F32 MRd, MR	32 Ra e, MRf	

MSUBF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 32-bit Floating-Point Subtraction with Parallel Move

Operands								
-	MRd	CL MF	A floating-point de the s	stination register (M ame register as MR	R0 to MR3) for the M a	ISUBF32 operation		
	MRe	CL	A floating-point so	urce register (MR0 t	o MR3) for the MSUI	BF32 operation		
	MRf	CL	A floating-point so	urce register (MR0 t	o MR3) for the MSUI	BF32 operation		
	MRa	CL MF	A floating-point de Ra cannot be the s	stination register (M ame register as MR	R0 to MR3) for the M d	1MOV32 operation		
	mem32	32- MN	bit memory location.	on accessed using d	irect or indirect addre	essing. Source for the		
Opcode	LSW: mmmm mm MSW: 0010 ff	mm mmmm Eee ddaa	mmmm addr					
Description	Subtract the floating-point MRd = MRe - MRa = [mem32	Subtract the contents of two floating-point registers and move from memory to a floating-point register. MRd = MRe - MRf; MRa = [mem32];						
Restrictions	The destinat MRa cannot	ion regis be the s	ter for the MSU ame register a	JBF32 and the N s MRd.	MOV32 must b	e unique. That is,		
Flags	This instruction modifies the following flags in the MSTF register:							
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	Yes	Yes	Yes	Yes		
	The MSTF re LUF = 1 i LVF = 1 i The MMOV3	egister fla if MSUBI f MSUBI 2 Instruc	ags are modifie F32 generates F32 generates ction will set the	ed as follows: an underflow co an overflow con e NF and ZF flag	ndition. dition. gs as follows:			
Pipeline	Both MSUBF	-32 and	MMOV32 com	plete in a single	cycle.			



Instruction Set

Example

NF = MRa(31); ZF = 0; if(MRa(30:23) == 0) { ZF = 1; NF = 0; }

See also MSUBF32 MRa, MRb, MRc MSUBF32 MRa, #16FHi, MRb MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRd, MRe, MRf ||MMOV32 mem32, MRa 32-bit Floating-Point Subtraction with Parallel Move

Operands							
	MRd	C	LA floating-point de	stination register (M	R0 to MR3) for the M	ASUBF32 operation	
	MRe	C	LA floating-point so	urce register (MR0 t	o MR3) for the MSU	BF32 operation	
	MRf	C	LA floating-point so	urce register (MR0 t	o MR3) for the MSU	BF32 operation	
	mem32	32	2-bit destination me	mory location for the	e MMOV32 operatior	ı	
	MRa	C	LA floating-point so	urce register (MR0 t	o MR3) for the MMC	V32 operation	
Opcode	LSW: mmmm MSW: 0110	mmmm mmmm ffee ddaa	a mmmm a addr				
Description	Subtract to register to MRd = MRe [mem32] =	he content memory. - MRf; MRa;	s of two floating	g-point registers	and move from	a floating-point	
Flags	This instruction modifies the following flags in the MSTF register:						
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	Yes	Yes	
	The MST	- reaister f	ilaas are modifi	ed as follows:			
	 LUF = LVF = 	1 if MSUE 1 if MSUE	F32 generates F32 generates	an underflow co an overflow cor	ondition. Idition.		
Pipeline	Both MSU	IBF32 and	MMOV32 com	plete in a single	cycle.		
Example							
See also	MSUBF32 MSUBF32 MSUBF32 MMPYF32	2 MRa, MR 2 MRa, #16 2 MRd, MR 2 MRa, MR	₹b, MRc 3FHi, MRb ₹e, MRf MMO ₹b, MRc MSL	V32 MRa, mem IBF32 MRd, MR	32 e, MRf		



MSWAPF MRa, MRb {, CNDF} Conditional Swap

Operands

Operatius						
	MRa	CLA	floating-poir	nt register (MR0 to MR3)		
	MRb	CLA	floating-poir	nt register (MR0 to MR3)		
	CNDF	Opti	onal conditio	n tested based on the MSTF fla	gs	
Opcode	LSW: 0000 MSW: 0111	0000 CNDF 1 1011 0000	bbaa 0000			
Description	Conditiona	I swap of N	IRa and M	Rb.		
	CNDF is o	ne of the fo	llowing co	nditions:		
	Encode ⁽¹⁾	CNDF	iowing oo	Description	MSTF Flags	Tested
	0000	NEQ		Not equal to zero	ZF == 0	
	0001	EQ		Equal to zero	ZF == 1	
	0010	GT		Greater than zero	ZF == 0 ANI	0 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 OR	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 ⁽²⁾		Unconditional with flag modification	None	
Flags	⁽¹⁾ Values n ⁽²⁾ This is th be modifi This instru	ot shown are r e default oper ed when a con	eserved. ation if no Cl nditional ope	NDF field is specified. This cond ration is executed. All other con-	ition will allow t ditions will not r	he ZF and NF flags to nodify these flags.
1 1495	Flag	TF		NF		LVF
	Modified	No	No	No	No	No

No flags affected

Pipeline

This is a single-cycle instruction.



Example	<pre>; X is an array ; and has len a; ; the array and ; ; Note: MCMPF3: ; ClalTask1:</pre>	Y of 32-bit float: elements. Find the d store it in Resu 2 and MSWAPF can b	ing-point values e maximum value in ult be replaced by MMAXF32
	MMOVI16 MUI16TOF32 MNOP MNOP MMOV32 LOOP MMOV32 MCMPF32 MSWAPF MADDF32 MCMPF32 MCMPF32 MNOP MNOP MNOP	MAR1,#_X MR0, @_len MR1, *MAR1[2]++ MR2, *MAR1[2]++ MR2, MR1 MR1, MR2, GT MR0, MR0, #-1.0 MR0 #0.0	<pre>; Start address ; Length of the array ; delay for MAR1 load ; delay for MAR1 load ; MR1 = X0 ; MR2 = next element ; Compare MR2 with MR1 ; MR1 = MAX(MR1, MR2) ; Decrement the counter ; Set/clear flags for MBCNDD</pre>
	MBCNDD MMOV32 MNOP MNOP MSTOP	LOOP, NEQ @_Result, MR1	<pre>; Branch if not equal to zero ; Always executed ; Always executed ; Always executed ; End of task</pre>

See also



MTESTTF CNDF Test MSTF Register Flag Condition

Operands

-	CNDF	condition to test based on MSTF flags
Opcode	LSW: 0000 0000 MSW: 0111 1111	0000 cndf 0100 0000

Description

Test the CLA floating-point condition and if true, set the MSTF[TF] flag. If the condition is false, clear the MSTF[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:

	0		
Encode (3)	CNDF	Description	MSTF 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 OR 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 ⁽⁴⁾	Unconditional with flag modification	None

(3) Values not shown are reserved.

(4) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF 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 MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	No	No	No	No
TF = 0;					

if (CNDF == true) TF = 1;

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

Pipeline

This is a single-cycle instruction.


www	.tı.cor	n

Example	; if (State ; RampSta ; else if (S ; CoastSt ; else ; SteadyS	== 0.1) te = RampState tate == 0.01) ate = CoastState tate = SteadyState	RAMPMASK COASTMASK STEADYMASK
	, _Cla1Task2: MMOV32 MCMPF32 MTESTTF MNOP MBCNDD MMOV32 MMOVX1 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32	MR0, @_State MR0, #0.1 MR0, #0.01 EQ _Skipl, NEQ MR1, @_RampState MR2, #RAMPMASK MR1, MR2 @_RampState, MR1	<pre>; Affects flags for 1st MBCNDD (A) ; Check used by 2nd MBCNDD (B) ; Store EQ flag in TF for 2nd MBCNDD (B) ; (A) If State != 0.1, go to Skipl ; Always executed ; Always executed ; Always executed ; Always executed ; Execute if (A) branch not taken ; end of task if (A) branch not taken</pre>
	_Skip1: MMOV32 MMOVXI MOR32 MMOV32 MMOV32 MMOV32 MMOV32 MMOV32 MSTOP _Skip2:	MR3, @_SteadyStat MR2, #STEADYMASK MR3, MR2 _Skip2, NTF MR1, @_CoastState MR2, #COASTMASK MR1, MR2 @_CoastState, MR1	e ; (B) if State != .01, go to Skip2 ; Always executed ; Always executed ; Always executed ; Execute if (B) branch not taken ; end of task if (B) branch not taken
	MMOV32 @ MSTOP	SteadyState, MR3	; Executed if (B) branch taken

See also



MUI16TOF32 MRa, mem16 Convert unsigned 16-bit integer to 32-bit floating-point value

	MRa	CL	A floating-point de	stination register (N	IR0 to MR3)		
	mem16	16	-bit source memor	y location			
Opcode	LSW: mmmm	mmmm mmmm	mmmm				
	MSW: 0111	0101 01aa	addr				
Description	When con to zero wh	verting F3	2 to I16/UI16 d 32TOI16R/UI1	ata format, the I 6R operation wil	MF32TOI16/UI16	operation trunca st (even) value.	tes
	MRa = UI1	6TOF32[mem	16];				
Flags	This instru	uction does	not affect any	flags:			
-	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a s	single-cycle	e instruction.				
Example							
See also	MF32TOI MF32TOL MF32TOL MF32TOL MI16TOF MI16TOF MU16TO	16 MRa, M 16R MRa, JI16 MRa, JI16R MRa J2 MRa, M 32 MRa, m F32 MRa,	Rb MRb MRb , MRb Rb em16 MRb				



MUI16TOF32 MRa, MRb Convert unsigned 16-bit integer to 32-bit floating-point value

	MRa	Cl	A floating-point de	estination register (M	R0 to MR3)		
	MRb	Cl	A floating-point so	ource register (MR0 t	o MR3)		
Opcode	LSW: 0000 MSW: 0111	0000 0000 1110 1110	bbaa 0000				
Description	Convert an unsigned 16-bit integer to a 32-bit floating-point value. When converting float32 to I16/UI16 data format, the MF32TOI16/UI16 operation truncates to zero whil the MF32TOI16R/UI16R operation will round to nearest (even) value.						
Flags	This instru	uction does	s not affect any	flags:			
	Flag	TF	ZF	NF	LUF	LVF	
	Modified	No	No	No	No	No	
Pipeline	This is a s	single-cycle	e instruction.				
Example	MMOVXI MR MUI16TOF3	1, #0x800F 2 MR0, MR1	; MR1(15:0) = ; MR0 = UI16 ; = 32783.0	= 32783 (0x800F) FOF32 (MR1(15:0) (0x47000F00))))		
See also	MF32TOI MF32TOI MF32TOI MF32TOI MI16TOF MI16TOF MUI16TO	16 MRa, M 16R MRa, JI16 MRa, JI16R MRa 32 MRa, M 32 MRa, m F32 MRa,	IRb MRb MRb a, MRb IRb iem16 mem16				

MUI32TOF32 MRa, mem32 Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value

	MRa	CLA flo	ating-point des	tination register (M	IR0 to MR3)			
	mem32	32-bit r	nemory location	n accessed using d	lirect or indirect addres	sing		
Opcode	LSW: mmmm mr MSW: 0111 0	nmm mmmm mmr 100 10aa ado	nm lr					
Description	MRa = UI32TC	OF32[mem32]	;					
Flags	This instruct	ion does not	t affect any f	lags:	1115	LVE		
	Modified	No	No	No	No	No		
Pipeline	This is a sing	gle-cycle ins	struction.					
Example	<pre>; Given x2, m2 and b2 are Uint32 numbers: ; ; x2 = Uint32(2) = 0x00000002 ; m2 = Uint32(1) = 0x00000001 ; b2 = Uint32(3) = 0x00000003 ; ; Calculate y2 = x2 * m2 + b2 ; ClalTask1: MUI32TOF32 MR0, @_m2</pre>							
See also	MF32TOI32 MF32TOUI3 MI32TOF32 MI32TOF32 MUI32TOF32	MRa, MRb 2 MRa, MRI MRa, mem MRa, MRb 2 MRa, MRb	b 32 b					



MUI32TOF32 MRa, MRb Convert Unsigned 32-bit Integer to 32-bit Floating-Point Value

	MRa	C	LA floating-point de	estination register (N	IR0 to MR3)	
	MRb	С	LA floating-point so	ource register (MR0	to MR3)	
Opcode	LSW: 0000	0000 000	0 bbaa			
	MSW: 0111	1101 110	0 0000			
Description	MRa = UI3	2TOF32 [M	Rb];			
Flags	This instru	uction doe	s not affect any	flags:		
	Flag	TF	ZF	NF	LUF	LVF
	Modified	No	No	No	No	No
Pipeline	This is a s	single-cycl	e instruction.			
Example	MMOVIZ MMOVXI	MR3, #0: MR3, #0:	x8000 ; MR3(31 x1111 ; MR3(15 ; MR3 = 2	:16) = 0x8000 :0) = 0x1111 2147488017		
	MUI32TOF3	2 MR3, MR	3 ; MR3 = 1	MUI32TOF32 (MR3) = 2147488017.0) (0x4F000011)
See also	MF32TOI MF32TOI MI32TOF MI32TOF MUI32TO	32 MRa, N JI32 MRa, 32 MRa, n 32 MRa, N 9F32 MRa,	/Rb MRb nem32 /Rb mem32			



Instruction Set

MXOR32 MRa, MRb, MRc Bitwise Exclusive Or

operanao								
	MRa	CL	A floating-point des	stination register (M	IR0 to MR3)			
	MRb	CL	A floating-point sou	urce register (MR0 t	to MR3)			
	MRc	CL	A floating-point sou	urce register (MR0 t	to MR3)			
Dpcode	LSW: 0000 MSW: 0111	LSW: 0000 000cc bbaa MSW: 0111 1100 1010 0000						
Description	Bitwise XC	DR of MRb	with MRc. 1:0) XOR MRc(3	1:0);				
lags	This instru	uction modi	fies the followir	ng flags in the M	ISTF register:			
	Flag	TF	ZF	NF	LUF	LVF		
	Modified	No	Yes	Yes	No	No		
ipeline	This is a s	ingle-cycle	instruction.					
Pipeline	if(MRa(31 This is a s	:0) == 0) sinale-cvcle	$\{ ZF = 1; \}$					
Example	MMOVIZ MR MMOVXI 1 MMOVIZ 1	0, #0x5555 MRO, #0xAAA MR1, #0x54	; MRO = 0x5 AA 32 ; MR1 = 0:	555AAAA x5432FEDC				
	MMOVXI	MR1, #0xFE	DC					
	; 0101 ; ; 0101 ; ; 0101 ; ; 0101 ; ; 1010 ; ; 1010 ; ; 1010 ;	XOR 0101 = XOR 0100 = XOR 0011 = XOR 0101 = XOR 1111 = XOR 1110 = XOR 1101 = XOR 1100 =	0000 (0) 0001 (1) 0110 (6) 0111 (7) 0101 (5) 0100 (4) 0111 (7) 0110 (6)					
	MXOR32	MR2, MR1, 1	MRO ; MR3 = 0:	x01675476				
See also	MAND32 MOR32 M	MRa, MRb IRa, MRb, I	, MRc MRc					



Appendix A CLA and CPU Arbitration

Typically, CLA activity is independent of the CPU activity. Under the circumstance where both the CLA and the CPU are attempting to access memory or a peripheral register within the same interface concurrently, an arbitration procedure will occur. This appendix describes this arbitration.

A.1 CLA and CPU Arbitration

Typically, CLA activity is independent of the CPU activity. Under the circumstance where both the CLA and the CPU are attempting to access memory or a peripheral register within the same interface concurrently, an arbitration procedure will occur. The one exception is the ADC result registers which do not create a conflict when read by both the CPU and the CLA simultaneously even if different addresses are accessed. Any combined accesses between the different interfaces, or where the CPU access is outside of the interface that the CLA is accessing do not create a conflict.

The interfaces that can have conflict arbitration are:

- CLA Message RAMs
- CLA Program Memory
- CLA Data RAMs

A.1.1 CLA Message RAMs

Message RAMs consist of two blocks. These blocks are for passing data between the main CPU and the CLA. No opcode fetches are allowed from the message RAMs. The two message RAMs have the following characteristics:

CLA to CPU Message RAM:

The following accesses are allowed:

- CPU reads
- CLA reads and writes
- CPU debug reads and writes
- The following accesses are ignored
- CPU writes
- Priority of accesses are (highest priority first):
- 1. CLA write
- 2. CPU debug write
- 3. CPU data read, program read, CPU debug read
- 4. CLA data read

CPU to CLA Message RAM:

The following accesses are allowed:

- CPU reads and writes
- CLA reads
- CPU debug reads and writes
- The following accesses are ignored
- CLA writes

Priority of accesses are (highest priority first):

- 1. CLA read
- 2. CPU data write, program write, CPU debug write
- 3. CPU data read, CPU debug read
- 4. CPU program read



CLA and CPU Arbitration

A.1.2 CLA Program Memory

The behavior of the program memory depends on the state of the MMEMCFG[PROGE] bit. This bit controls whether the memory is mapped to CLA space or CPU space.

• MMEMCFG[PROGE] == 0

In this case the memory is mapped to the CPU. The CLA will be halted and no tasks shoud be incoming.

- Any CLA fetch will be treated as an illegal opcode condition as described in Section 3.4. This
 condition will not occur if the proper procedure is followed to map the program memory.
- CLA reads and writes cannot occur

The memory block behaves as any normal SRAM block mapped to CPU memory space.

Priroty of accesses are (highest priority first):

- 1. CPU data write, program write, debug write
- 2. CPU data read, program read, debug read
- 3. CPU fetch, program read

MMEMCFG[PROGE] == 1

In this case the memory block is mapped to CLA space. The CPU can only make debug accesses.

- CLA reads and writes cannot occur
- CLA fetches are allowed
- CPU fetches return 0 which is an illegal opcode and will cause an ITRAP interrupt.
- CPU data reads and program reads return 0
- CPU data writes and program writes are ignored

Priroty of accesses are (highest priority first):

- 1. CLA fetch
- 2. CPU debug write
- 3. CPU debug read
- **NOTE:** Because the CLA fetch has higher priority than CPU debug reads, it is possible for the CLA to permanently block debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug. To avoid this issue, the program memory will return all 0x0000 for CPU debug reads (ignore writes) when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access can be performed.





A.1.3 CLA Data Memory

There are two independent data memory blocks. The behavior of the data memory depends on the state of the MMEMCFG[RAM0E] and MMEMCFG[RAM1E] bits. These bits determine whether the memory blocks are mapped to CLA space or CPU space.

• MMEMCFG[RAMxE] == 0

In this case the memory block is mapped to the CPU.

- CLA fetches cannot occur to this block.
- CLA reads return 0
- CLA writes are ignored
- The memory block behaves as any normal SARAM block mapped to the CPU memory space.

Priroty of accesses are (highest priority first):

- 1. CPU data write, program write, debug write
- 2. CPU data read, program read, debug read
- 3. CPU fetch, program read

• MMEMCFG[RAMxE] == 1

In this case th ememory block is mapped to CLA space. The CPU can only make debug accesses.

- CLA fetches cannot occur to this block.
- CLA read and CLA writes are allowed.
- CPU fetches return 0
- CPU data reads and program reads return 0
- CPU data writes and program writes are ignored

Priroty of accesses are (highest priority first):

- 1. CLA write
- 2. CPU debug write
- 3. CPU debug read
- 4. CLA read

A.1.4 Peripheral Registers (ePWM, HRPWM, Comparator)

Accesses to the registers follow these rules:

- If both the CPU and CLA request access at the same time, then the CLA will have priority and the main CPU is stalled.
- If a CPU access is in progress and another CPU access is pending, then the CLA will have priority over the pending CPU access. In this case the CLA access will begin when the current CPU access completes.
- While a CPU access is in progress any incoming CLA access will be stalled.
- While a CLA access is in progress any incoming CPU access will be stalled.
- A CPU write operation has priority over a CPU read operation.
- A CLA write operation has priority over a CLA read operation.
- If the CPU is performing a read-modify-write operation and the CLA performs a write to the same location, the CLA write may be lost if the operation occurs in-between the CPU read and write. For this reason, you should not mix CPU and CLA accesses to same location.

Appendix B Revision History

This document has been revised because of the following technical change(s).

Table 34. Revisions to this Document

Section 4.3.3 For bits 15-12, value 0010, changed ePWM5 to ePWM4 is the input for interrupt task 4. (EPWM4_INT) for bit 15-12 description	Location	Edits, Deletes, Additions
	Section 4.3.3	For bits 15-12, value 0010, changed ePWM5 to ePWM4 is the input for interrupt task 4. (EPWM4_INT) for bit 15-12 description.

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
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2010, Texas Instruments Incorporated