MICROCHIP

Section 2. MCU

HIGHLIGHTS

This section of the manual contains the following topics:

2.1 Introduction 2-2
2.2 Architecture Overview 2-3
2.3 PIC32MX CPU Details 2-6
2.4 Special Considerations when Writing to CP0 Registers 2-11
2.5 Architecture Release 2 Details 2-12
2.6 Split CPU bus 2-12
2.7 Internal system busses 2-13
2.8 Set/Clear/Invert 2-13
2.9 ALU Status Bits 2-14
2.10 Interrupt and Exception Mechanism 2-14
2.11 Programming Model 2-15
2.12 CP0 Registers 2-22
2.13 MIPS16e™ Execution 2-58
2.14 Memory Model 2-58
2.15 CPU Instructions, Grouped By Function 2-60
2.16 CPU Initialization 2-64
2.17 Effects of a Reset 2-65
2.18 Related Application Notes 2-67
2.19 Revision History 2-68

© 2008 Microchip Technology Inc.

2.1 INTRODUCTION

The PIC32MX Microcontroller Unit (MCU) is a complex system-on-a-chip that is based on a M4K™ core from MIPS® Technologies. M4K™ is a state-of-the-art 32-bit, low-power, RISC processor core with the enhanced MIPS32® Release 2 Instruction Set Architecture. This chapter provides an overview of the CPU features and system architecture of the PIC32MX family of microcontrollers.

Key Features

- Up to 1.5 DMIPS/MHz of performance
- Programmable prefetch cache memory to enhance execution from Flash memory
- 16-bit Instruction mode (MIPS16e) for compact code
- Vectored interrupt controller with 63 priority levels
- Programmable User and Kernel modes of operation
- Atomic bit manipulations on peripheral registers (Single cycle)
- Multiply-Divide unit with a maximum issue rate of one 32 × 16 multiply per clock
- High speed Microchip ICD port with hardware-based non-intrusive data monitoring and application data streaming functions
- EJTAG debug port allows extensive third party debug, programming and test tools support
- Instruction controlled power management modes
- Five stage piplined instruction execution
- Internal Code protection to help protect intellectual property

Related MIPS® Documentation

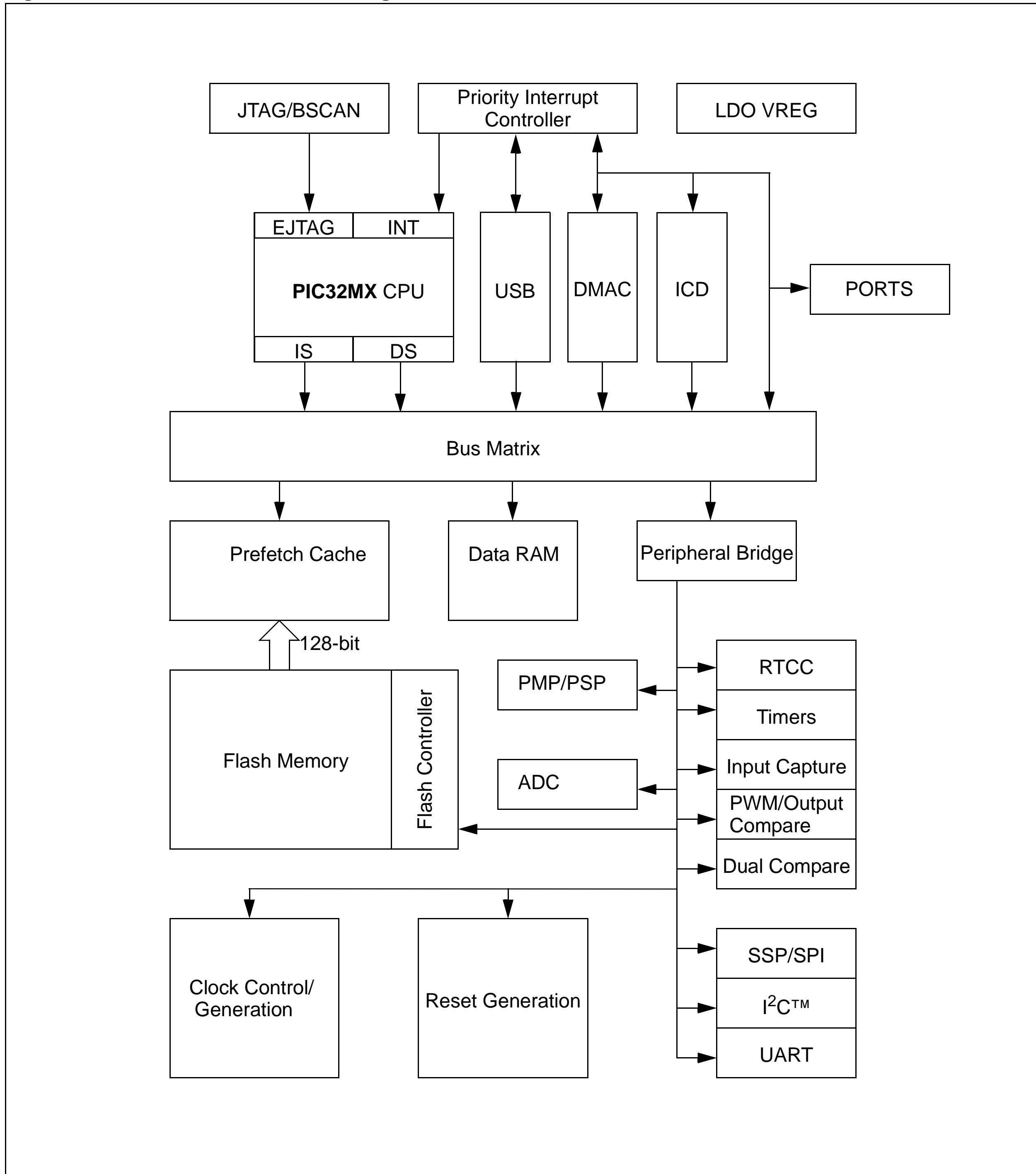
- MIPS M4K™ Software User's Manual – MD00249-2B-M4K-SUM
- MIPS® Instruction Set – MD00086-2B-MIPS32BIS-AFP
- MIPS16e™ – MD00076-2B-MIPS1632-AFP
- MIPS32® Privileged Resource Architecture – MD00090-2B-MIPS32PRA-AFP

© 2008 Microchip Technology Inc.

2.2 ARCHITECTURE OVERVIEW

The PIC32MX family processors are complex systems-on-a-chip that contain many features. Included in all processors of the PIC32MX family is a high-performance RISC CPU, which can be programmed in 32-bit and 16-bit modes, and even mixed modes. The PIC32MX MCU contains a high-performance interrupt controller, DMA controller, USB controller, in-circuit debugger, high performance switching matrix for high-speed data accesses to the peripherals, on-chip data RAM memory that holds data and programs. The unique prefetch cache and prefetch buffer for the Flash memory, which hides the latency of the Flash, gives zero Wait state equivalent performance.

Figure 2-1: PIC32MX MCU Block Diagram



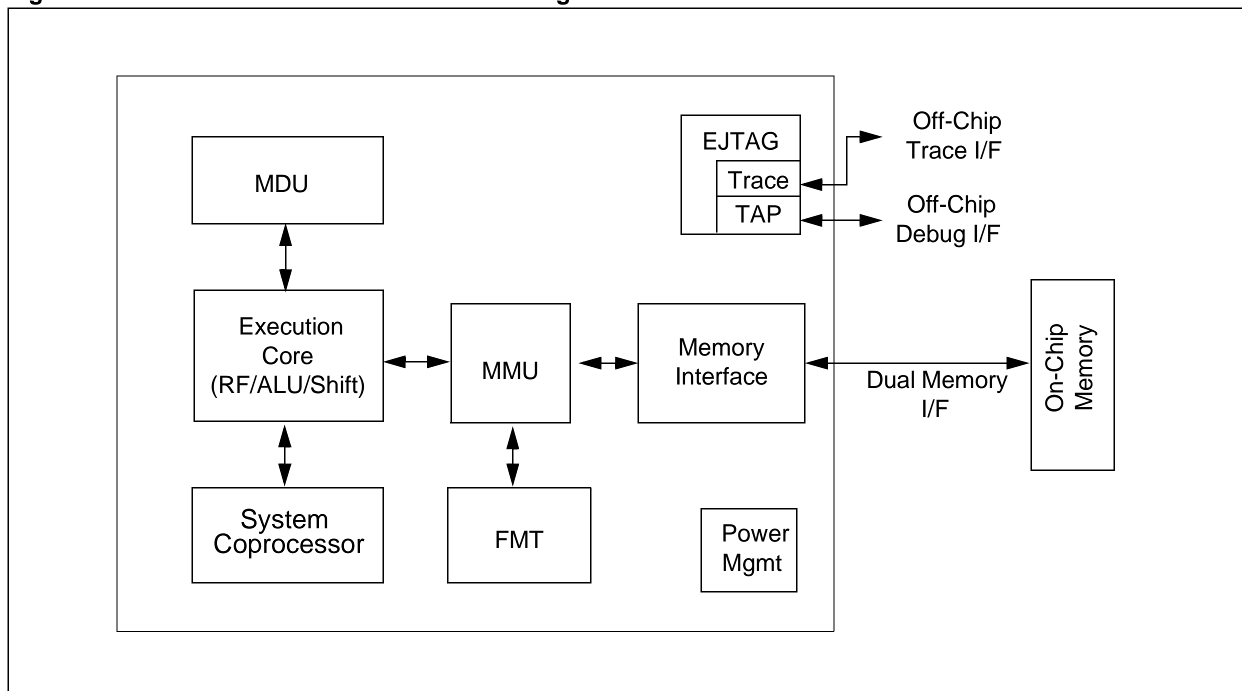
2

MCU

There are two internal buses in the chip to connect all the peripherals. The main peripheral bus connects most of the peripheral units to the bus matrix through a peripheral bridge. There is also a high-speed peripheral bridge that connects the interrupt controller DMA controller, in-circuit debugger, and USB peripherals. The heart of the PIC32MX MCU is the M4K CPU core. The CPU performs operations under program control. Instructions are fetched by the CPU, decoded and executed synchronously. Instructions exist in either the Program Flash memory or Data RAM memory.

The PIC32MX CPU is based on a load/store architecture and performs most operations on a set of internal registers. Specific load and store instructions are used to move data between these internal registers and the outside world.

Figure 2-2: M4K™ Processor Core Block Diagram



2.2.1 Busses

There are two separate busses on the PIC32MX MCU. One bus is responsible for the fetching of instructions to the CPU, and the other is the data path for load and store instructions. Both the instruction, or I-side bus, and the data, or D-side bus, are connected to the bus matrix unit. The bus matrix is a switch that allows multiple accesses to occur concurrently in a system. The bus matrix allows simultaneous accesses between different bus masters that are not attempting accesses to the same target. The bus matrix serializes accesses between different masters to the same target through an arbitration algorithm.

Since the CPU has two different data paths to the bus matrix, the CPU is effectively two different bus masters to the system. When running from Flash memory, load and store operations to SRAM and the internal peripherals will occur in parallel to instruction fetches from Flash memory.

In addition to the CPU, there are three other bus masters in the PIC32MX MCU – the DMA controller, In-Circuit-Debugger Unit, and the USB controller.

  © 2008 Microchip Technology Inc.

2.2.2 Introduction to the Programming Model

The PIC32MX processor has the following features:

- 5-stage pipeline
- 32-bit Address and Data Paths
- DSP-like Multiply-add and multiply-subtract instructions (MADD, MADDU, MSUB, MSUBU)
- Targeted multiply instruction (MUL)
- Zero and One detect instructions (CLZ, CLO)
- Wait instruction (WAIT)
- Conditional move instructions (MOVZ, MOVN)
- Implements MIPS32 Enhanced Architecture (Release 2)
- Vectored interrupts
- Programmable exception vector base
- Atomic interrupt enable/disable
- General Purpose Register (GPR) shadow sets
- Bit field manipulation instructions
- MIPS16e Application Specific Extension improves code density
- Special PC-relative instructions for efficient loading of addresses and constants
- Data type conversion instructions (ZEB, SEB, ZEH, SEH)
- Compact jumps (JRC, JALRC)
- Stack frame set-up and tear down “macro” instructions (SAVE and RESTORE)
- Memory Management Unit with simple Fixed Mapping Translation (FMT)
- Processor to/from Coprocessor register data transfers
- Direct memory to/from Coprocessor register data transfers
- Performance-optimized Multiply-Divide Unit (High-performance build-time option)
- Maximum issue rate of one 32 × 16 multiply per clock
- Maximum issue rate of one 32 × 32 multiply every other clock
- Early-in divide control – 11 to 34 clock latency
- Low-Power mode (triggered by WAIT instruction)
- Software breakpoints via the SDBBP instruction

2.2.3 Core Timer

The PIC32MX architecture includes a core timer that is available to application programs. This timer is implemented in the form of two co-processor registers–the Count register (CP0_COUNT), and the Compare register (CP0_COMPARE). The Count register is incremented every two system clock (SYSCLK) cycles. The incrementing of Count can be optionally suspended during Debug mode. The Compare register is used to cause a timer interrupt if desired. An interrupt is generated when the Compare register matches the Count register. An interrupt is taken only if it is enabled in the interrupt controller.

For more information on the core timer, see **Section 2.12. “CP0 Registers”** and **Section 8. “Interrupts.”**

© 2008 Microchip Technology Inc.

2.3 PIC32MX CPU DETAILS

2.3.1 Pipeline Stages

The pipeline consists of five stages:

- Instruction (I) Stage
- Execution (E) Stage
- Memory (M) Stage
- Align (A) Stage
- Writeback (W) Stage

2.3.1.1 I Stage – Instruction Fetch

During I stage:

- An instruction is fetched from the instruction SRAM.
- MIPS16e instructions are converted into MIPS32-like instructions.

2.3.1.2 E Stage – Execution

During E stage:

- Operands are fetched from the register file.
- Operands from the M and A stage are bypassed to this stage.
- The Arithmetic Logic Unit (ALU) begins the arithmetic or logical operation for register-to-register instructions.
- The ALU calculates the data virtual address for load and store instructions and the MMU performs the fixed virtual-to-physical address translation.
- The ALU determines whether the branch condition is true and calculates the virtual branch target address for branch instructions.
- Instruction logic selects an instruction address and the MMU performs the fixed virtual-to-physical address translation.
- All multiply divide operations begin in this stage.

2.3.1.3 M Stage – Memory Fetch

During M stage:

- The arithmetic or logic ALU operation completes.
- The data SRAM access is performed for load and store instructions.
- A 16 × 16 or 32 × 16 MUL operation completes in the array and stalls for one clock in the M stage to complete the carry-propagate-add in the M stage.
- A 32 × 32 MUL operation stalls for two clocks in the M stage to complete the second cycle of the array and the carry-propagate-add in the M stage.
- Multiply and divide calculations proceed in the MDU. If the calculation completes before the IU moves the instruction past the M stage, then the MDU holds the result in a temporary register until the IU moves the instructions to the A stage (and it is consequently known that it won’t be killed).

2.3.1.4 A Stage – Align

During A stage:

- A separate aligner aligns loaded data with its word boundary.
- A MUL operation makes the result available for writeback. The actual register writeback is performed in the W stage.
- From this stage, load data or a result from the MDU are available in the E stage for bypassing.

 © 2008 Microchip Technology Inc.

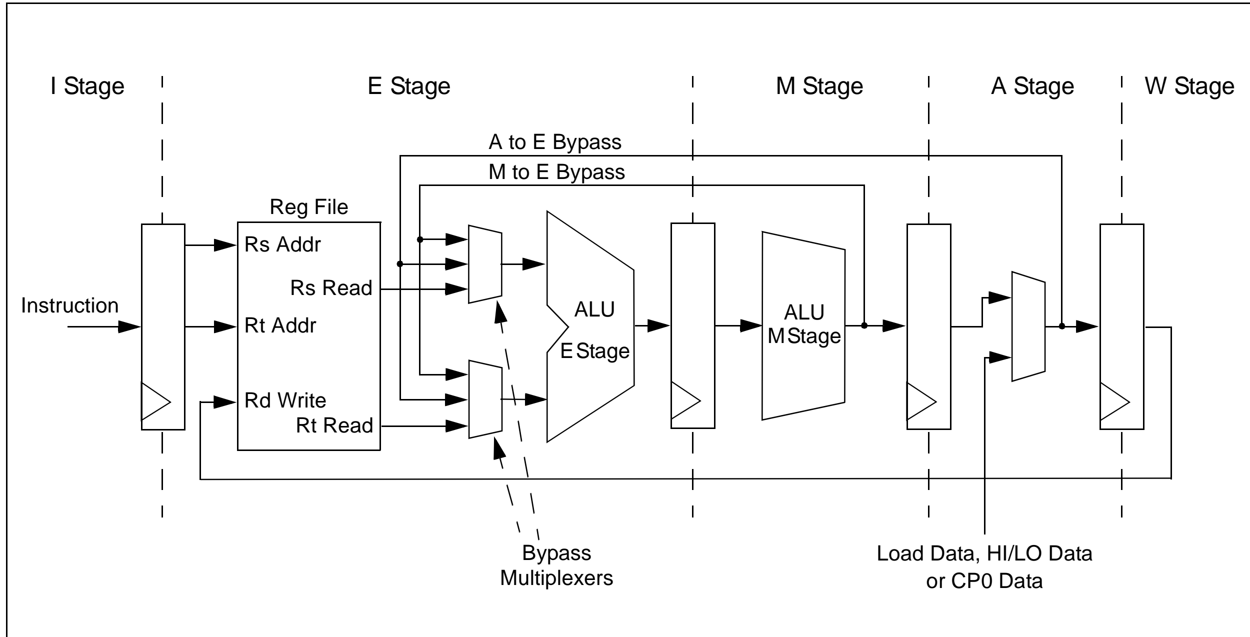
2.3.1.5 W Stage – Writeback

During W stage:

For register-to-register or load instructions, the result is written back to the register file.

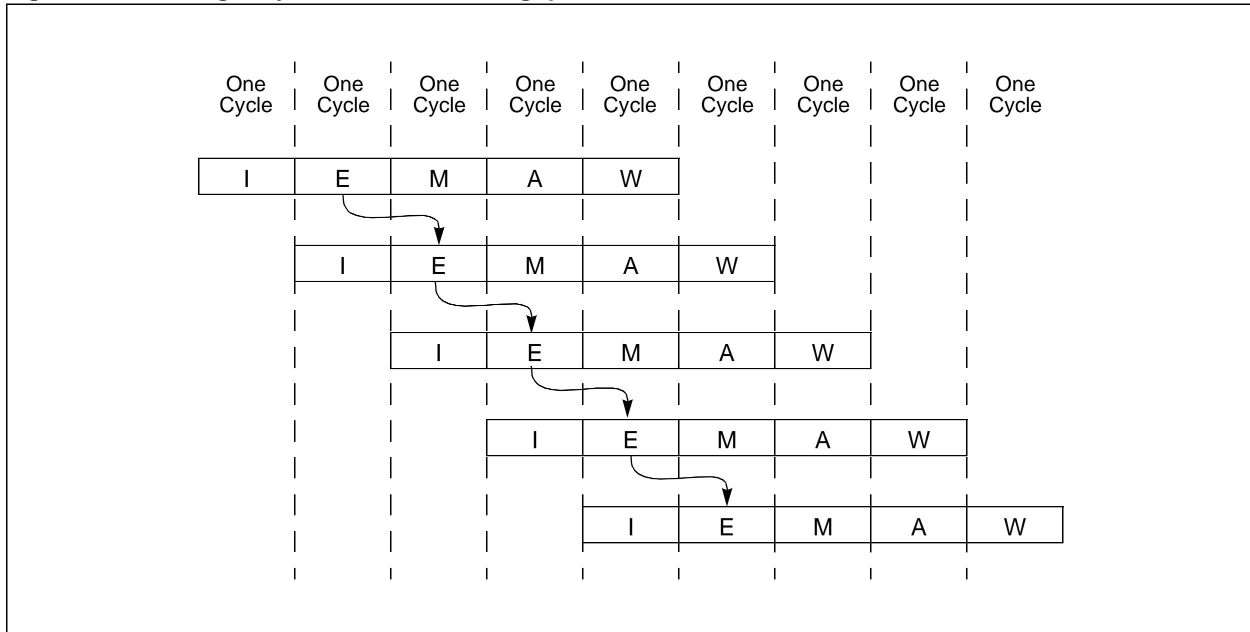
A M4K core implements a “Bypass” mechanism that allows the result of an operation to be sent directly to the instruction that needs it without having to write the result to the register and then read it back.

Figure 2-3: Simplified PIC32MX CPU Pipeline



The results of using instruction pipelining in the PIC32MX core is a fast, single-cycle instruction execution environment.

Figure 2-4: Single-Cycle Execution Throughput



© 2008 Microchip Technology Inc.

2.3.2 Execution Unit

The PIC32MX Execution Unit is responsible for carrying out the processing of most of the instructions of the MIPS instruction set. The Execution Unit provides single-cycle throughput for most instructions by means of pipelined execution. Pipelined execution, sometimes referred to as “pipelining”, is where complex operations are broken into smaller pieces called stages. Operation stages are executed over multiple clock cycles.

The Execution Unit contains the following features:

- 32-bit adder used for calculating the data address
- Address unit for calculating the next instruction address
- Logic for branch determination and branch target address calculation
- Load aligner
- Bypass multiplexers used to avoid stalls when executing instructions streams where data producing instructions are followed closely by consumers of their results
- Leading Zero/One detect unit for implementing the `CLZ` and `CLO` instructions
- Arithmetic Logic Unit (ALU) for performing bitwise logical operations
- Shifter and Store Aligner

2.3.3 MDU

The Multiply/Divide unit performs multiply and divide operations. The MDU consists of a 32 × 16 multiplier, result-accumulation registers (HI and LO), multiply and divide state machines, and all multiplexers and control logic required to perform these functions. The high-performance, pipelined MDU supports execution of a 16 × 16 or 32 × 16 multiply operation every clock cycle; 32 × 32 multiply operations can be issued every other clock cycle. Appropriate interlocks are implemented to stall the issue of back-to-back 32 × 32 multiply operations. Divide operations are implemented with a simple 1 bit per clock iterative algorithm and require 35 clock cycles in worst case to complete. Early-in to the algorithm detects sign extension of the dividend, if it is actual size is 24, 16 or 8 bit. the divider will skip 7, 15, or 23 of the 32 iterations. An attempt to issue a subsequent `MDU` instruction while a divide is still active causes a pipeline stall until the divide operation is completed.

The M4K implements an additional multiply instruction, `MUL`, which specifies that lower 32-bits of the multiply result be placed in the register file instead of the HI/LO register pair. By avoiding the explicit move from LO (`MFLO`) instruction, required when using the LO register, and by supporting multiple destination registers, the throughput of multiply-intensive operations is increased. Two instructions, multiply-add (`MADD/MADDU`) and multiply-subtract (`MSUB`/`MSUBU`), are used to perform the multiply-add and multiply-subtract operations. The `MADD` instruction multiplies two numbers and then adds the product to the current contents of the HI and LO registers. Similarly, the MSUB instruction multiplies two operands and then subtracts the product from the HI and LO registers. The `MADD`/`MADDU` and `MSUB`/`MSUBU` operations are commonly used in Digital Signal Processor (DSP) algorithms.

2.3.4 Shadow Register Sets

The PIC32MX processor implements a copy of the General Purpose Registers (GPR) for use by high-priority interrupts. This extra bank of registers is known as a shadow register set. When a high-priority interrupt occurs the processor automatically switches to the shadow register set without software intervention. This reduces overhead in the interrupt handler and reduces effective latency.

The shadow register set is controlled by registers located in the System Coprocessor (CP0) as well as the interrupt controller hardware located outside of the CPU core.

For more information on shadow register sets, see the XREF Interrupt chapter.

© 2008 Microchip Technology Inc.

2.3.5 Pipeline Interlock Handling

Smooth pipeline flow is interrupted when an instruction in a pipeline stage can not advance due to a data dependency or a similar external condition. Pipeline interruptions are handled entirely in hardware. These dependencies, are referred to as *interlocks*. At each cycle, interlock conditions are checked for all active instructions. An instruction that depends on the result of a previous instruction is an example of an interlock condition.

In general, MIPS processors support two types of hardware interlocks:

- Stalls

  Stalls are resolved by halting the entire pipeline. All instructions currently executing in each pipeline stage are affected by a stall.
- Slips

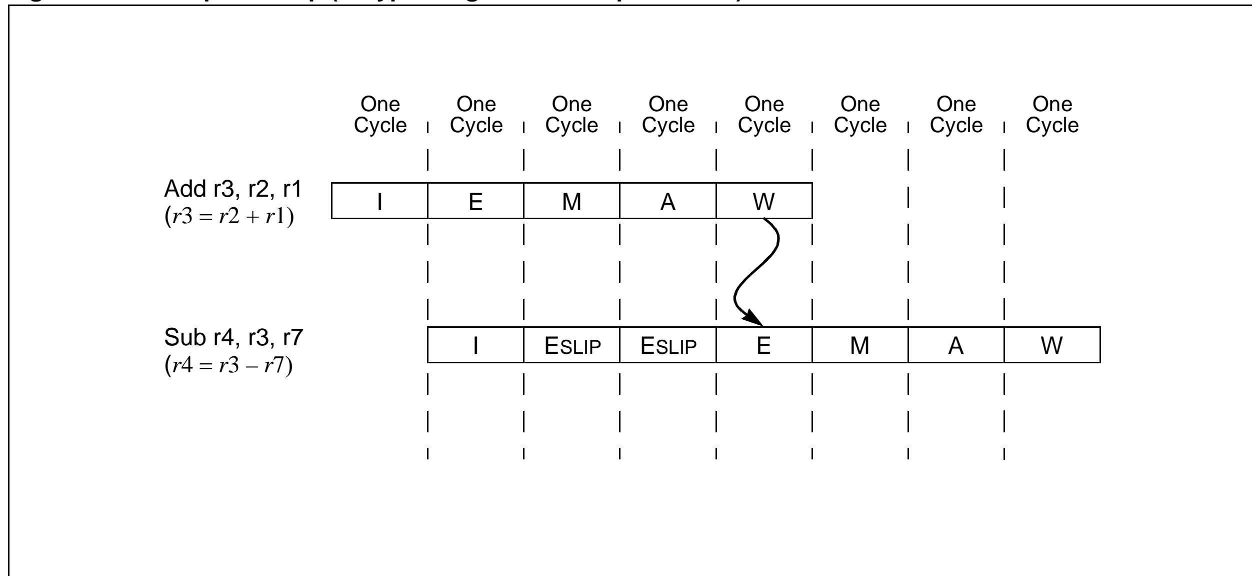
  Slips allow one part of the pipeline to advance while another part of the pipeline is held static.

In the PIC32MX processor core, all interlocks are handled as slips. These slips are minimized by grabbing results from other pipeline stages by using a method called register bypassing, which is described below.

Note:	To illustrate the concept of a pipeline slip, the following example is what would happen if the PIC32MX core did not implement register bypassing.

As shown in Figure 2-5, the sub instruction has a source operand dependency on register r3 with the previous add instruction. The sub instruction slips by two clocks waiting until the result of the add is written back to register r3. This slipping does *not* occur on the PIC32MX family of processors.

Figure 2-5: Pipeline Slip (If Bypassing Was Not Implemented)



© 2008 Microchip Technology Inc.

2.3.6 Register Bypassing

As mentioned previously, the PIC32MX processor implements a mechanism called register bypassing that helps reduce pipeline slips during execution. When an instruction is in the E stage of the pipeline, the operands must be available for that instruction to continue. If an instruction has a source operand that is computed from another instruction in the execution pipeline, register bypassing allows a shortcut to get the source operands directly from the pipeline. An instruction in the E stage can retrieve a source operand from another instruction that is executing in either the M stage or the A stage of the pipeline. As seen in Figure 2-6, a sequence of three instructions with interdependencies does not slip at all during execution. This example uses both A to E, and M to E register bypassing. Figure 2-7 shows the operation of a load instruction utilizing A to E bypassing. Since the result of load instructions are not available until the A pipeline stage, M to E bypassing is not needed.

The performance benefit of register bypassing is that instruction throughput is increased to the rate of one instruction per clock for ALU operations, even in the presence of register dependencies.

Figure 2-6: IU Pipeline M to E Bypass

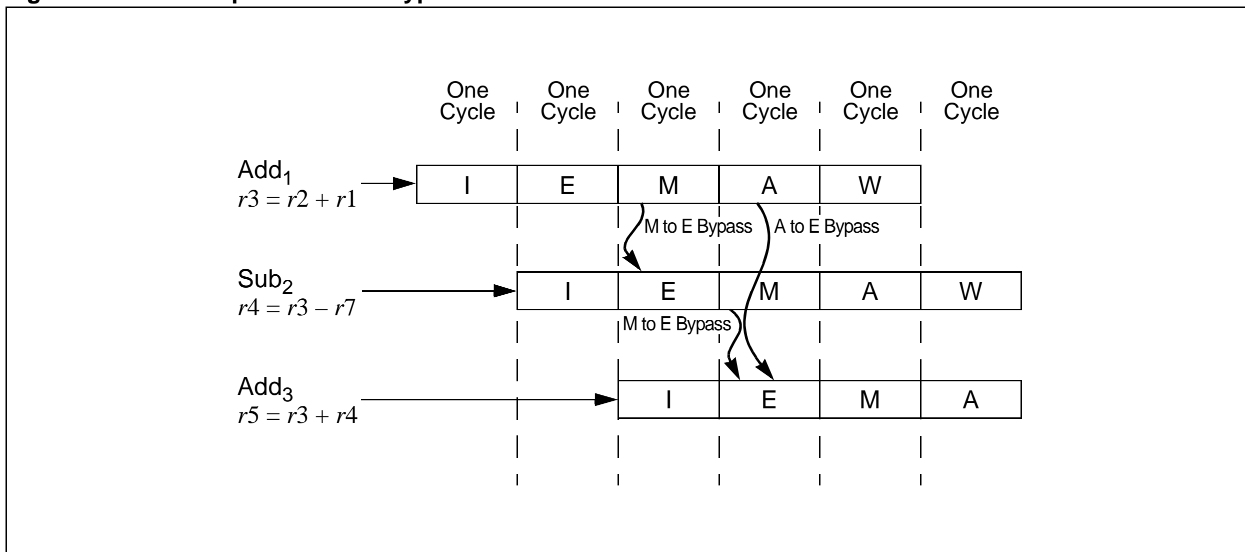
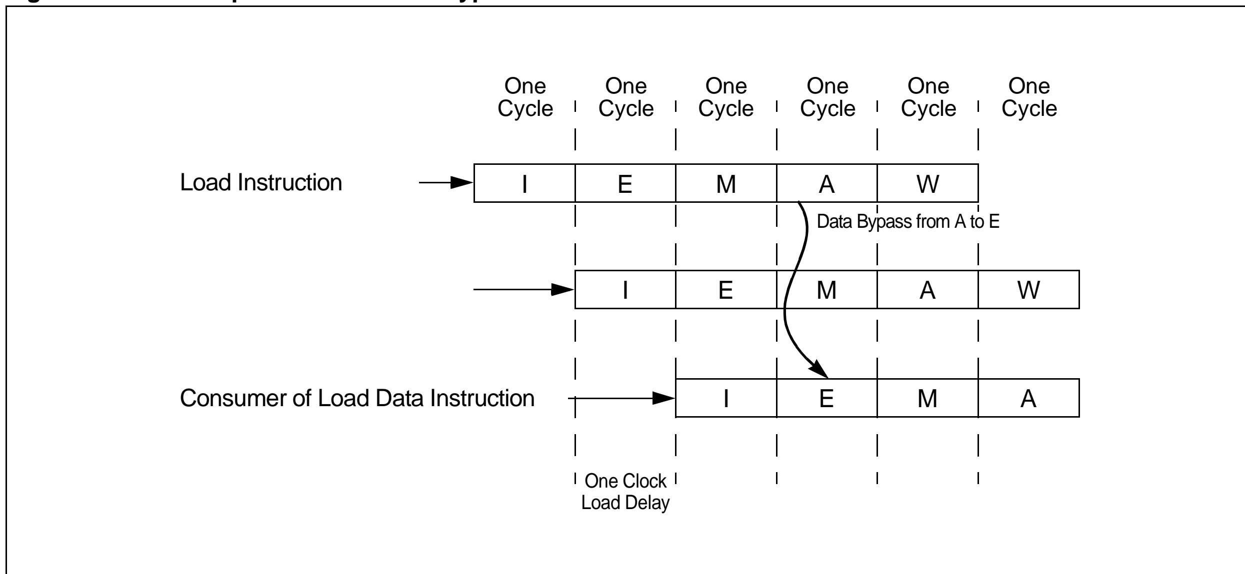


Figure 2-7: IU Pipeline A to E Data Bypass



 © 2008 Microchip Technology Inc.

2.4 SPECIAL CONSIDERATIONS WHEN WRITING TO CP0 REGISTERS

In general, the PIC32MX core ensures that instructions are executed following a fully sequential program model. Each instruction in the program sees the results of the previous instruction. There are some deviations to this model. These deviations are referred to as *hazards*.

In privileged software, there are two different types of hazards:

- Execution Hazards
- Instruction Hazards

2.4.0.1 Execution Hazards

Execution hazards are those created by the execution of one instruction, and seen by the execution of another instruction. Table 2-1 lists execution hazards.

Table 2-1: Execution Hazards

Producer	=	Consumer	Hazard On	Spacing (Instructions)
MTC0	=	Coprocessor instruction execution depends on the new value of $Status_{CU}$	$Status_{CU}$	1
MTC0	=	ERET	EPC DEPC ErrorEPC	1
MTC0	=	ERET	Status	0
MTC0, EI, DI	=	Interrupted Instruction	$Status_{IE}$	1
MTC0	=	Interrupted Instruction	$Cause_{IP}$	3
MTC0	=	RDPGPR WRPGPR	$SRSCtl_{PSS}$	1
MTC0	=	Instruction not seeing a Timer Interrupt	Compare update that clears Timer Interrupt	4
MTC0	=	Instruction affected by change	Any other CP0 register	2

2.4.0.2 Instruction Hazards

Instruction hazards are those created by the execution of one instruction, and seen by the instruction fetch of another instruction. Table 2-2 lists instruction hazards.

Table 2-2: Instruction Hazards

Producer	=	Consumer	Hazard On
MTC0	=	Instruction fetch seeing the new value (including a change to ERL followed by an instruction fetch from the useg segment)	Status

2.5 ARCHITECTURE RELEASE 2 DETAILS

The PIC32MX CPU utilizes Release 2 of the MIPS 32-bit architecture. The PIC32MX CPU implements the following Release 2 features:

- Vectored interrupts using and external-to-core interrupt controller

  Provide the ability to vector interrupts directly to a handler for that interrupt.
- Programmable exception vector base

  Allows the base address of the exception vectors to be moved for exceptions that occur when $Status_{BEV}$ is '0'. This allows any system to place the exception vectors in memory that is appropriate to the system environment.
- Atomic interrupt enable/disable

  Two instructions have been added to atomically enable or disable interrupts, and return the previous value of the *Status* register.
- The ability to disable the *Count* register for highly power-sensitive applications.
- GPR shadow registers

  Provides the addition of GPR shadow registers and the ability to bind these registers to a vectored interrupt or exception.
- Field, Rotate and Shuffle instructions

  Add additional capability in processing bit fields in registers.
- Explicit hazard management

  Provides a set of instructions to explicitly manage hazards, in place of the cycle-based SSNOP method of dealing with hazards.

2.6 SPLIT CPU BUS

The PIC32MX CPU core has two distinct busses to help improve system performance over a single-bus system. This improvement is achieved through parallelism. Load and store operations occur at the same time as instruction fetches. The two busses are known as the I-side bus which is used for feeding instructions into the CPU, and the D-side bus used for data transfers.

The CPU fetches instructions during the I pipeline stage. A fetch is issued to the I-side bus and is handled by the bus matrix unit. Depending on the address, the BMX will do one of the following:

- Forward the fetch request to the Prefetch Cache Unit
- Forward the fetch request to the DRM unit or
- Cause an exception

Instruction fetches always use the I-side bus independent of the addresses being fetched. The BMX decides what action to perform for each fetch request based on the address and the values in the BMX registers. (See BMX chapter).

The D-side bus processes all load and store operations executed by the CPU. When a load or store instruction is executed the request is routed to the BMX by the D-side bus. This operation occurs during the M pipeline stage and is routed to one of several targets devices:

- Data Ram
- Prefetch Cache/Flash Memory
- Fast Peripheral Bus (Interrupt controller, DMA, Debug unit, USB, GPIO Ports)
- General Peripheral Bus (UART, SPI, Flash Controller, EPMP/EPSP, TRCC Timers, Input Capture, PWM/Output Compare, ADC, Dual Compare, I^2C, Clock SIB, and Reset SIB)

 © 2008 Microchip Technology Inc.

2.7 INTERNAL SYSTEM BUSSES

The PIC32MX processor internal busses connect the peripherals to the bus matrix unit. The bus matrix routes bus accesses from 5 different initiators to a set of targets utilizing several data paths throughout the chip to help eliminate performance bottlenecks.

Some of the paths that the bus matrix uses serve a dedicated purpose, while others are shared between several targets.

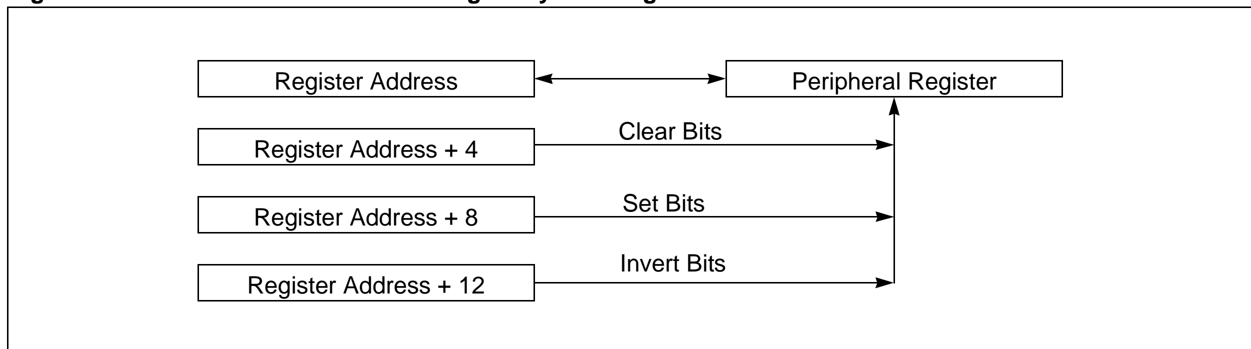
The data RAM and Flash memory read paths are dedicated paths, allowing low-latency access to the memory resources without being delayed by peripheral bus activity. The high-bandwidth peripherals are placed on a high-speed bus. These include the Interrupt controller, debug unit, DMA engine, and the USB host/peripheral unit.

Peripherals that do not require high-bandwidth are located on a separate peripheral bus to save power.

2.8 SET/CLEAR/INVERT

To provide single-cycle bit operations on peripherals, the registers in the peripheral units can be accessed in three different ways depending on peripheral addresses. Each register has four different addresses. Although the four different addresses appear as different registers, they are really just four different methods to address the same physical register.

Figure 2-8: Four Addresses for a Single Physical Register



The base register address provides normal Read/Write access, the other three provide special write-only functions.

1. Normal access
2. Set bit atomic RMW access
3. Clear bit atomic RMW access
4. Invert bit atomic RMW access

Peripheral reads must occur from the base address of each peripheral register. Reading from a set/clear/invert address has an undefined meaning, and may be different for each peripheral.

Writing to the base address writes an entire value to the peripheral register. All bits are written. For example, assume a register contains 0xaaaa5555 before a write of 0x000000ff. After the write, the register will contain 0x000000ff (assuming that all bits are R/W bits).

Writing to the Set address for any peripheral register causes only the bits written as '1's to be set in the destination register. For example, assume that a register contains 0xaaaa5555 before a write of 0x000000ff to the set register address. After the write to the Set register address, the value of the peripheral register will contain 0xaaaa55ff.

Writing to the Clear address for any peripheral register causes only the bits written as '1's to be cleared to '0's in the destination register. For example, assume that a register contains 0xaaaa5555 before a write of 0x000000ff to the Clear register address. After the write to the Clear register address, the value of the peripheral register will contain 0xaaaa5500.

Writing to the Invert address for any peripheral register causes only the bits written as '1's to be inverted, or toggled, in the destination register. For example, assume that a register contains 0xaaaa5555 before a write of 0x000000ff to the invert register address. After the write to the Invert register, the value of the peripheral register will contain 0xaaaa55aa.

2.9 ALU STATUS BITS

Unlike most other PIC® microcontrollers, the PIC32MX Processor does not use STATUS register flags. Condition flags are used on many processors to help perform decision making operations during program execution. Flags are set based on the results of comparison operations or some arithmetic operations. Conditional branch instructions on these machines then make decisions based on the values of the single set of condition codes.

The PIC32MX processor, instead, uses instructions that perform a comparison and stores a flag or value into a General Purpose Register. A conditional branch is then executed with this general purpose register used as an operand.

2.10 INTERRUPT AND EXCEPTION MECHANISM

The PIC32MX family of processors implement an efficient and flexible interrupt and exception handling mechanism. Interrupts and exceptions both behave similarly in that the current instruction flow is changed temporarily to execute special procedures to handle an interrupt or exception. The difference between the two is that interrupts are usually a result of normal operation, and exceptions are a result of error conditions such as bus errors.

When an interrupt or exception occurs, the processor does the following:

1. The PC of the next instruction to execute after the handler returns is saved into a coprocessor register.
2. Cause register is updated to reflect the reason for exception or interrupt
3. Status EXL or ERL is set to cause Kernel mode execution
4. Handler PC is calculated from EBASE and SPACING values
5. Processor starts execution from new PC

This is a simplified overview of the interrupt and exception mechanism. See **Section 8. "Interrupts"** for more information regarding interrupt and exception handling.

 © 2008 Microchip Technology Inc.

2.11 PROGRAMMING MODEL

The PIC32MX family of processors is designed to be used with a high-level language such as the C programming language. It supports several data types and uses simple but flexible addressing modes needed for a high-level language. There are 32 General Purpose Registers and two special registers for multiplying and dividing.

There are three different formats for the machine language instructions on the PIC32MX processor:

- immediate or I-type CPU instructions
- jump or J-type CPU instructions and
- registered or R-type CPU instructions

Most operations are performed in registers. The register type CPU instructions have three operands; two source operands and a destination operand.

Having three operands and a large register set allows assembly language programmers and compilers to use the CPU resources efficiently. This creates faster and smaller programs by allowing intermediate results to stay in registers rather than constantly moving data to and from memory.

The immediate format instructions have an immediate operand, a source operand and a destination operand. The jump instructions have a 26-bit relative instruction offset field that is used to calculate the jump destination.

2.11.1 CPU Instruction Formats

A CPU instruction is a single 32-bit aligned word. The CPU instruction formats are shown below:

- Immediate (see Figure 2-9)
- Jump (see Figure 2-10)
- Register (see Figure 2-11)

Table 2-3 describes the fields used in these instructions.

Table 2-3: CPU Instruction Format Fields

Field	Description
opcode	6-bit primary operation code
rd	5-bit specifier for the destination register
rs	5-bit specifier for the source register
rt	5-bit specifier for the target (source/destination) register or used to specify functions within the primary opcode REGIMM
immediate	16-bit signed immediate used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement
instr_index	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address
sa	5-bit shift amount
function	6-bit function field used to specify functions within the primary opcode SPECIAL

Figure 2-9: Immediate (I-Type) CPU Instruction Format

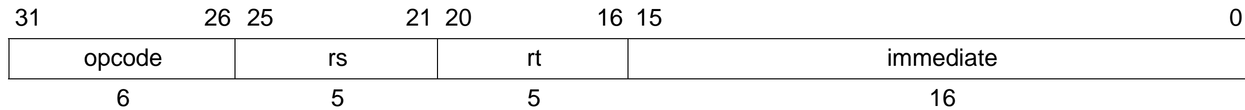


Figure 2-10: Jump (J-Type) CPU Instruction Format

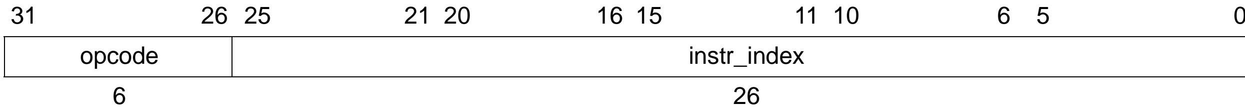
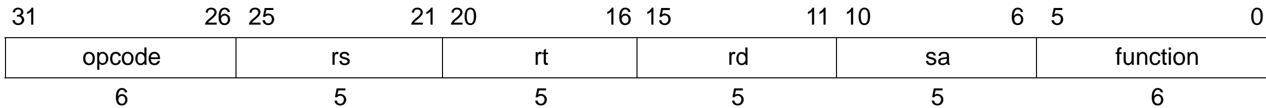


Figure 2-11: Register (R-Type) CPU Instruction Format



2.11.2 CPU Registers

The PIC32MX architecture defines the following CPU registers:

- 32 32-bit General Purpose Registers (GPRs)
- 2 special purpose registers to hold the results of integer multiply, divide, and multiply-accumulate operations (HI and LO)
- a special purpose program counter (PC), which is affected only indirectly by certain instructions – it is not an architecturally visible register.

2.11.2.1 CPU General Purpose Registers

Two of the CPU General Purpose Registers have assigned functions:

- r0

  r0 is hard-wired to a value of '0', and can be used as the target register for any instruction the result of which will be discarded. r0 can also be used as a source when a '0' value is needed.
- r31

  r31 is the destination register used by JAL, BLTZAL, BLTZALL, BGEZAL, and BGEZALL, without being explicitly specified in the instruction word. Otherwise r31 is used as a normal register.

The remaining registers are available for general purpose use.

 Preliminary © 2008 Microchip Technology Inc.

2.11.2.2 Register Conventions

Although most of the registers in the PIC32MX architecture are designated as General Purpose Registers, there are some recommended uses of the registers for correct software operation with high-level languages such as the Microchip C compiler.

Table 2-4: Register Conventions

CPU Register	Symbolic Register	Usage
r0	zero	Always 0[1]
r1	at	Assembler Temporary
r2 - r3	v0-v1	Function Return Values
r4 - r7	a0-a3	Function Arguments
r8 - r15	t0-t7	Temporary – Caller does not need to preserve contents
r16 - r23	s0-s7	Saved Temporary – Caller must preserve contents
r24 - r25	t8 - t9	Temporary – Caller does not need to preserve contents
r26 - r27	k0 - k1	Kernel temporary – Used for interrupt and exception handling
r28	gp	Global Pointer – Used for fast-access common data
r29	sp	Stack Pointer – Software stack
r30	s8 or fp	Saved Temporary – Caller must preserve contents *OR* Frame Pointer – Pointer to procedure frame on stack
r31	ra	Return Address[1]

Note 1: Hardware enforced, not just convention.

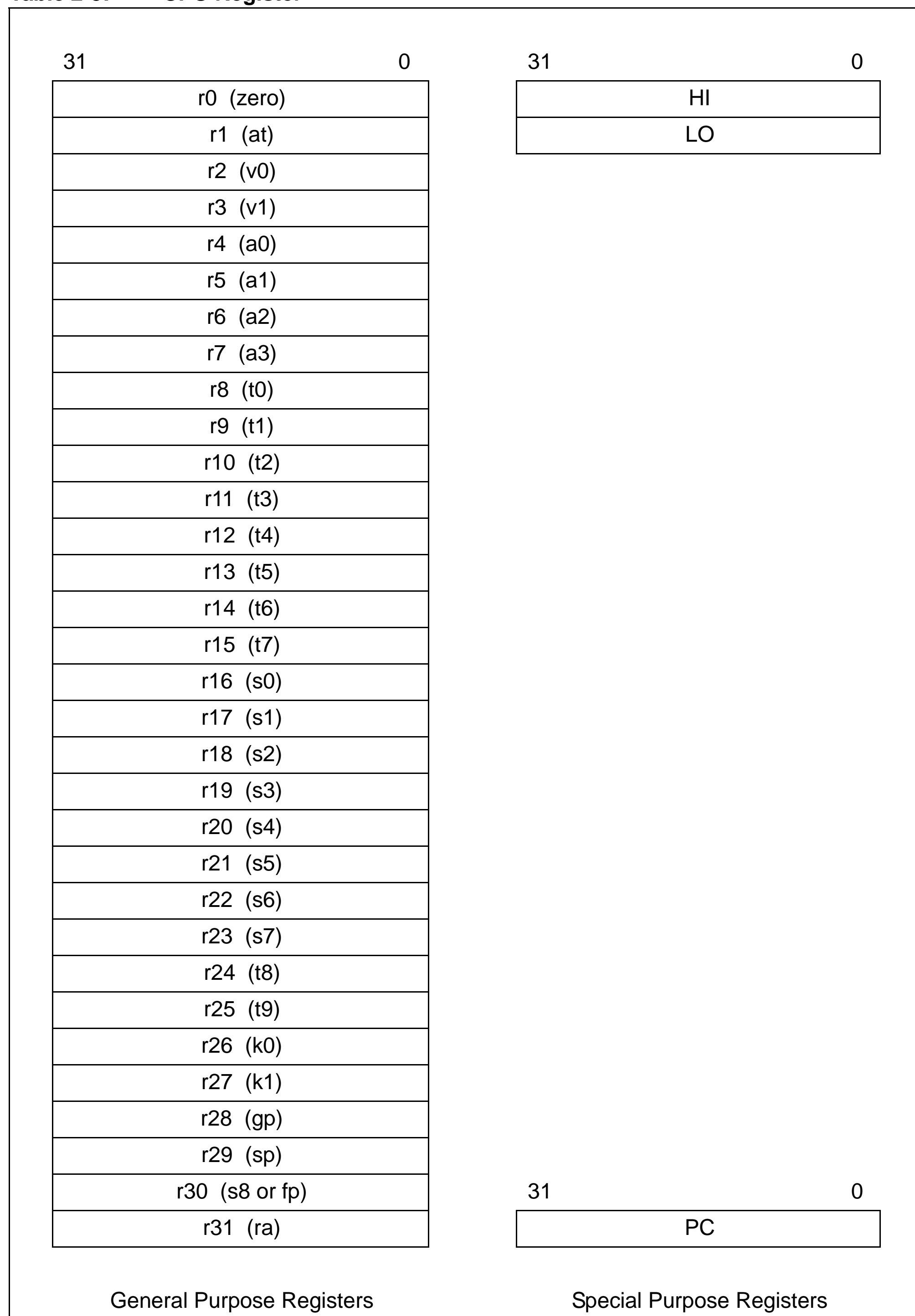
2.11.2.3 CPU Special Purpose Registers

The CPU contains three special purpose registers:

- PC – Program Counter register
- HI – Multiply and Divide register higher result
- LO – Multiply and Divide register lower result
  - During a multiply operation, the HI and LO registers store the product of integer multiply.
  - During a multiply-add or multiply-subtract operation, the HI and LO registers store the result of the integer multiply-add or multiply-subtract.
  - During a division, the HI and LO registers store the quotient (in LO) and remainder (in HI) of integer divide.
  - During a multiply-accumulate, the HI and LO registers store the accumulated result of the operation.

Figure 2-12 shows the layout of the CPU registers.

Table 2-5: CPU Register



 Preliminary © 2008 Microchip Technology Inc.

Table 2-6: MIPS16e Register Usage

MIPS16e Register Encoding	32-Bit MIPS Register Encoding	Symbolic Name	Description
0	16	s0	General Purpose Register
1	17	s1	General Purpose Register
2	2	v0	General Purpose Register
3	3	v1	General Purpose Register
4	4	a0	General Purpose Register
5	5	a1	General Purpose Register
6	6	a2	General Purpose Register
7	7	a3	General Purpose Register
N/A	24	t8	MIPS16e Condition Code register; implicitly referenced by the `BTEQZ`, `BTNEZ`, `CMP`, `CMPI`, `SLT`, `SLTU`, `SLTI`, and `SLTIU` instructions
N/A	29	sp	Stack Pointer register
N/A	31	ra	Return Address register

Table 2-7: MIPS16e Special Registers

Symbolic Name	Purpose
PC	Program counter. PC-relative `Add` and `Load` instructions can access this register as an operand.
HI	Contains high-order word of multiply or divide result.
LO	Contains low-order word of multiply or divide result.

2.11.3 How to implement a stack/MIPS calling conventions

The PIC32MX CPU does not have hardware stacks. Instead, the processor relies on software to provide this functionality. Since the hardware does not perform stack operations itself, a convention must exist for all software within a system to use the same mechanism. For example, a stack can grow either toward lower address, or grow toward higher addresses. If one piece of software assumes that the stack grows toward lower address, and calls a routine that assumes that the stack grows toward higher address, the stack would become corrupted.

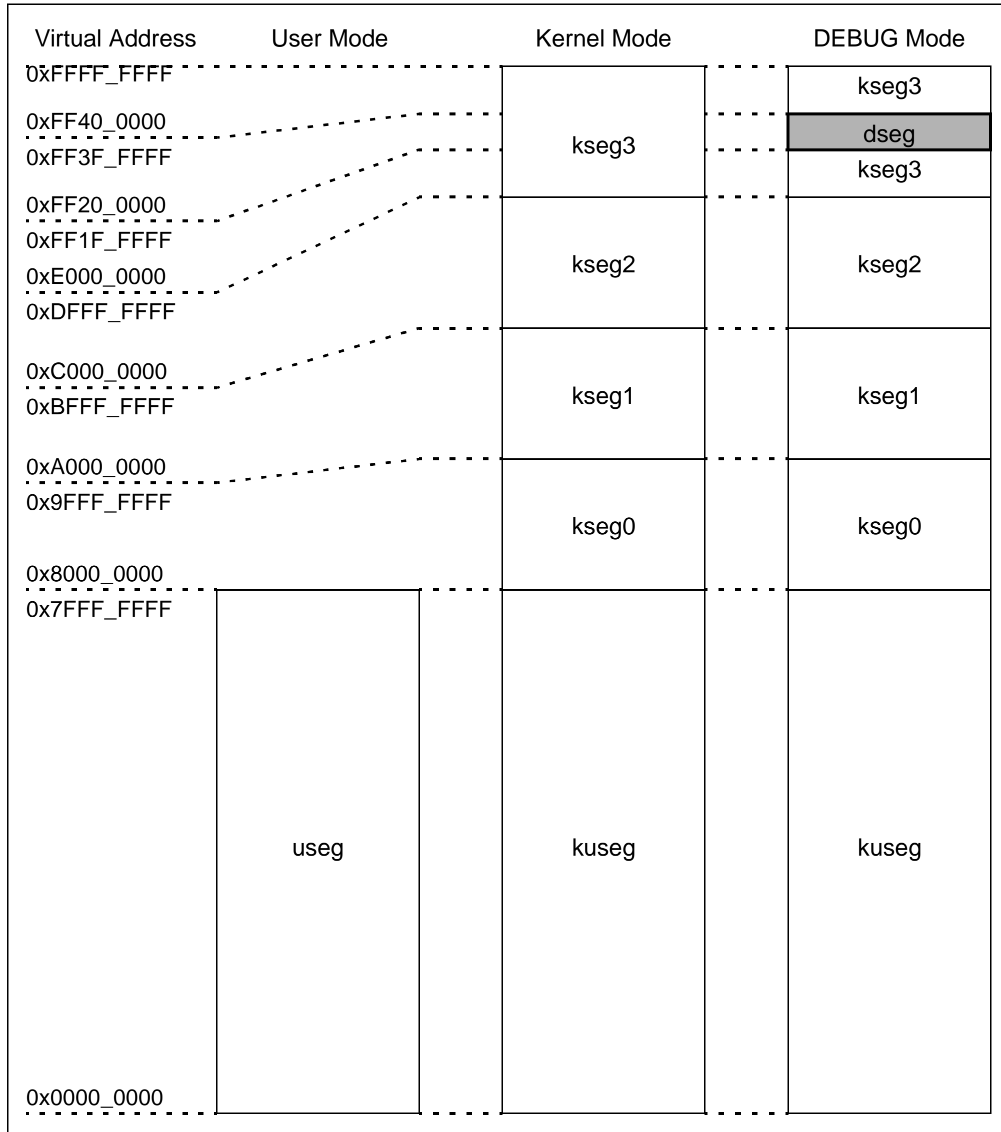
Using a system-wide calling convention prevents this problem from occurring. The Microchip C compiler assumes the stack grows toward lower addresses.

2.11.4 Processor Modes

There are two operational modes and one special mode of execution in the PIC32MX family CPUs; User mode, Kernel mode and DEBUG mode. The processor starts execution in Kernel mode, and if desired, can stay in Kernel mode for normal operation. User mode is an optional mode that allows a system designer to partition code between privileged and un-privileged software. DEBUG mode is normally only used by a debugger or monitor.

One of the main differences between the modes of operation is the memory addresses that software is allowed to access. Peripherals are not accessible in User mode. Figure 2-12 shows the different memory maps for each mode. For more information on the processor's memory map, see **Section 3. "Memory Organization"**.

Figure 2-12: CPU Modes



 Preliminary © 2008 Microchip Technology Inc.

2.11.4.1 Kernel Mode

In order to access many of the hardware resources, the processor must be operating in Kernel mode. Kernel mode gives software access to the entire address space of the processor as well as access to privileged instructions.

The processor operates in Kernel mode when the DM bit in the DEBUG register is '0' and the STATUS register contains one, or more, of the following values:

UM = 0   ERL = 1   EXL = 1

When a non-debug exception is detected, EXL or ERL will be set and the processor will enter Kernel mode. At the end of the exception handler routine, an Exception Return (ERET) instruction is generally executed. The ERET instruction jumps to the Exception PC (EPC or ErrorPC depending on the exception), clears ERL, and clears EXL if ERL= 0.

If UM = 1 the processor will return to User mode after returning from the exception when ERL and EXL are cleared back to '0'.

2.11.4.2 User Mode

When executing in User mode, software is restricted to use a subset of the processor's resources. In many cases it is desirable to keep application-level code running in User mode where if an error occurs it can be contained and not be allowed to affect the Kernel mode code.

Applications can access Kernel mode functions through controlled interfaces such as the SYSCALL mechanism.

As seen in Figure 2-12, User mode software has access to the USEG memory area.

To operate in User mode, the STATUS register must contain each the following bit values:

UM = 1   EXL = 0   ERL = 0

2.11.4.3 DEBUG Mode

DEBUG mode is a special mode of the processor normally only used by debuggers and system monitors. DEBUG mode is entered through a debug exception and has access to all the Kernel mode resources as well as special hardware resources used to debug applications.

The processor is in DEBUG mode when the DM bit in the DEBUG register is '1'.

DEBUG mode is normally exited by executing a DERET instruction from the debug handler.

2.12 CP0 REGISTERS

The PIC32MX uses a special register interface to communicate status and control information between system software and the CPU. This interface is called Coprocessor 0. The features of the CPU that are visible through Coprocessor 0 are core timer, interrupt and exception control, virtual memory configuration, shadow register set control, processor identification, and debugger control. System software accesses the registers in CP0 using coprocessor instructions such as MFC0 and MTC0. Table 2-8 describes the CP0 registers found on the PIC32MX MCU.

Table 2-8: CP0 Registers

Register Number	Register Name	Function
0-6	Reserved	Reserved in the PIC32MX core
7	HWREna	Enables access via the `RDHWR` instruction to selected hardware registers in Non-privileged mode
8	BadVAddr	Reports the address for the most recent address-related exception
9	Count	Processor cycle count
10	Reserved	Reserved in the PIC32MX core
11	Compare	Timer interrupt control
12	Status/ IntCtl/ SRSCtl/ SRSMap	Processor status and control; interrupt control; and shadow set control
13	Cause	Cause of last exception
14	EPC	Program counter at last exception
15	PRId/ EBASE/	Processor identification and revision; exception base address
16	Config/ Config1/ Config2/ Config3	Configuration registers
17-22	Reserved	Reserved in the PIC32MX core
23	Debug/ Debug2/	Debug control/exception status and EJTAG trace control
24	DEPC	Program counter at last debug exception
25-29	Reserved	Reserved in the PIC32MX core
30	ErrorEPC	Program counter at last error
31	DeSAVE	Debug handler scratchpad register

 © 2008 Microchip Technology Inc.

2.12.1 HWREna Register (CP0 Register 7, Select 0)

HWREna contains a bit mask that determines which hardware registers are accessible via the RDHWR instruction.

Register 2-1: HWREna: Hardware Accessibility Register; CP0 Register 7, Select 0

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 15							bit 8

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	MASK<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-4 **Reserved:** Write ‘0’; returns ‘0’ on read

bit 3-0 **MASK<3:0>:** Bit Mask bits
1 = Access is enabled to corresponding hardware register
0 = Access is disabled
Each bit in this field enables access by the RDHWR instruction to a particular hardware register (which may not be an actual register). See the RDHWR instruction for a list of valid hardware registers.

2.12.2 BadVAddr Register (CP0 Register 8, Select 0)

BadVAddr is a read-only register that captures the most recent virtual address that caused an address error exception. Address errors are caused by executing load, store, or fetch operations from unaligned addresses, and also by trying to access Kernel mode addresses from User mode.

BadVAddr does not capture address information for bus errors, because they are not addressing errors.

Register 2-2: BadVAddr: Bad Virtual Address Register; CP0 Register 8, Select 0

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<31:24>							
bit 31							bit 24

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<23:16>							
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<15:8>							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **BadVAddr<31:0>:** Bad Virtual Address bits
Captures the virtual address that caused the most recent address error exception.

 © 2008 Microchip Technology Inc.

2.12.3 COUNT Register (CP0 Register 9, Select 0)

COUNT acts as a timer, incrementing at a constant rate, whether or not an instruction is executed, retired, or any forward progress is made through the pipeline. The counter increments every other clock, if the DC bit in the CAUSE register is '0'.

COUNT can be written for functional or diagnostic purposes, including at Reset or to synchronize processors.

By writing the CountDM bit in DEBUG register, it is possible to control whether COUNT continues to increment while the processor is in DEBUG mode.

Register 2-3: COUNT: Interval Counter Register; CP0 Register 9, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-0 **COUNT<31:0>:** Interval Counter bits
This value is incremented every other clock cycle.

© 2008 Microchip Technology Inc.

2.12.4 COMPARE Register (CP0 Register 11, Select 0)

COMPARE acts in conjunction with COUNT to implement a timer and timer interrupt function. COMPARE maintains a stable value and does not change on its own.

When the value of COUNT equals the value of COMPARE, the CPU asserts an interrupt signal to the system interrupt controller. This signal will remain asserted until COMPARE is written.

Register 2-4: COMPARE: Interval Count Compare Register; CP0 Register 11, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **COMPARE<31:0>:** Interval Count Compare Value bits

 © 2008 Microchip Technology Inc.

2.12.5 STATUS Register (CP0 Register 12, Select 0)

STATUS is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. Fields of this register combine to create operating modes for the processor.

2.12.5.0.1 **Interrupt Enable**

Interrupts are enabled when all of the following conditions are true:

IE = 1 EXL = 0 ERL = 0 DM = 0

If these conditions are met, then the settings of the IPL bits enable the interrupts.

2.12.5.0.2 **Operating Modes**

If the DM bit in the Debug register is ‘1’, then the processor is in DEBUG mode; otherwise, the processor is in either Kernel or User mode.

The following CPU STATUS register bit settings determine User or Kernel mode:

Table 2-9: CPU Status Bits that Determine Processor Mode

User Mode (requires *all* of the following bits and values)	UM = 1	EXL = 0	ERL = 0
Kernal Mode (requires *one* or more of the following bit values)	UM = 0	EXL = 1	ERL = 1

Note:	The STATUS register CU bits <31:28> control coprocessor accessibility. If any coprocessor is unusable, then an instruction that accesses it generates an exception.

Register 2-5: STATUS: Status Register; CP0 Register 12, Select 0

R-0	R-0	R-0	R/W-x	R/W-0(1)	r-x	R/W-x	r-0
CU3	CU2	CU1	CU0	RP	FR	RE	—
bit 31							bit 24

r-0	R/W-1	r-0	R/W-0	R/W-0	r-0	r-0	r-0
—	BEV	Reserved	SR	NMI	—	—	—
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
IPL<15:10>						R<9:8>	
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	UM	—	ERL	EXL	IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **CU3:** Coprocessor 3 Usable bit
Controls access to Coprocessor 3
COP3 is not supported. This bit cannot be written and will read as ‘0’

bit 30 **CU2:** Coprocessor 2 Usable bit
Controls access to Coprocessor 2.
COP2 is not supported. This bit cannot be written and will read as ‘0’

bit 29 **CU1:** Coprocessor 1 Usable bit
Controls access to Coprocessor 1
COP1 is not supported. This bit cannot be written and will read as ‘0’

bit 28 **CU0:** Coprocessor 0 Usable bit
Controls access to Coprocessor 0
0 = access not allowed
1 = access allowed
Coprocessor 0 is always usable when the processor is running in Kernel mode, independent of the state of the CU0 bit.

bit 27 **RP:** Reduced Powerbit
Enables reduced power mode

bit 26 **FR:** FR bit
Reserved on PIC32MX processors

bit 25 **RE:** Used to enable reverse-endian memory references while the processor is running in User mode
0 = User mode uses configured endianness
1 = User mode uses reversed endianness
Neither DEBUG mode nor Kernel mode nor Supervisor mode references are affected by the state of this bit.

bit 24:23 **R<24:23>:** Reserved. Ignored on write and read as ‘0’.

 © 2008 Microchip Technology Inc.

Register 2-5: STATUS: Status Register; CP0 Register 12, Select 0 (Continued)

bit 22 **BEV:** Control bit. Controls the location of exception vectors.
`0` = Normal
`1` = Bootstrap

bit 21 **Reserved**

bit 20 **SR:** Soft Reset bit
Indicates that the entry through the Reset exception vector was due to a Soft Reset.
`0` = Not Soft Reset (NMI or Reset)
`1` = Soft Reset
Software can only write a ‘`0`’ to this bit to clear it and cannot force a `0-1` transition.

bit 19 **NMI:** Soft Reset bit
Indicates that the entry through the reset exception vector was due to an NMI.
`0` = Not NMI (Soft Reset or Reset)
`1` = NMI
Software can only write a ‘`0`’ to this bit to clear it and cannot force a `0-1` transition.

bit 18 **R:** Reserved. ignored on write and read as ‘`0`’.

bit 17 **R:** Reserved. ignored on write and read as ‘`0`’.

bit 16 **R:** Reserved. ignored on write and read as ‘`0`’.

bit 15-10 **IPL<15:10>:** Interrupt Priority Level bits
This field is the encoded (0..63) value of the current IPL. An interrupt will be signaled only if the requested IPL is higher than this value

bit 9-8 **R<9:8>:** Reserved
These bits are writable, but have no effect on the interrupt system.

bit 7-5 **R<7:5>:** Reserved. Ignored on write and read as ‘`0`’

bit 4 **UM:**
This bit denotes the base operating mode of the processor. On the encoding of this bit is:
`0` = Base mode in Kernal mode
`1` = Base mode is User mode
Note: The processor can also be in Kernel mode if ERL or EXL is set, regardless of the state of the UM bit.

bit 3 **R:** Reserved. Ignored on write and read as ‘`0`’

bit 2 **ERL:** Error Level bit
Set by the processor when a Reset, Soft Reset, NMI or Cache Error exception are taken.
`0` = Normal level
`1` = Error level
When ERL is set:
- Processor is running in Kernel mode
- Interrupts are disabled
- `ERET` instruction will use the return address held in ErrorEPC instead of EPC
- Lower 2^{29} bytes of kuseg are treated as an unmapped and uncached region. This allows main memory to be accessed in the presence of cache errors. The operation of the processor is *undefined* if the ERL bit is set while the processor is executing instructions from kuseg.

bit 1 **EXL:** Exception Level bit
Set by the processor when any exception other than Reset, Soft Reset, or NMI exceptions is taken.
`0` = Normal level
`1` = Exception level

<u>When EXL is set:</u>
- Processor is running in Kernel Mode
- Interrupts are disabled

EPC, CauseBD and SRSCtl will not be updated if another exception is taken.

Register 2-5: STATUS: Status Register; CP0 Register 12, Select 0 (Continued)

bit 0	**IE:** Interrupt Enable bit Acts as the master enable for software and hardware interrupts: `0` = Interrupts are disabled `1` = Interrupts are enabled This bit may be modified separately via the `DI` and `EI` instructions

 © 2008 Microchip Technology Inc.

2.12.6 IntctI: Interrupt Control Register (CP0 Register 12, Select 1)

The Intctl register controls the vector spacing of the PIC32MX architecture.

Register 2-6: IntctI: Interrupt Control Register; CP0 Register 12, Select 1

R-0	R-0	R-0	R-0	R-0	R-0	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	VS<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
VS<7:5>			—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **R:** Reserved

bit 28-26 **R:** Reserved

bit 25-10 **Reserved:** Write ‘0’; ignore read
Must be written as ‘0’; returns ‘0’ on read.

bit 9-5 **VS<9:5>:** Vector Spacing bits
This field specifies the spacing between each interrupt vector.

Encoding	Spacing Between Vectors (hex)	Spacing Between Vectors (decimal)
16#00	16#000 0x	0
16#01	16#020	32
16#02	16#040	64
16#04	16#080	128
16#08	16#100	256
16#10	16#200	512

All other values are reserved. The operation of the processor is *undefined* if a reserved value is written to this field.

bit 4-0 **Unimplemented:** Read as ‘0’
Must be written as ‘0’; returns ‘0’ on read.

2.12.7 SRSCtl Register (CP0 Register 12, Select 2)

The SRSCtl register controls the operation of GPR shadow sets in the processor.

Table 2-10: Sources for New $SRSCtl_{CSS}$ on an Exception or Interrupt

Exception Type	Condition	$SRSCtl_{CSS}$ Source	Comment
Exception	All	$SRSCtl_{ESS}$	
Non-Vectored Interrupt	$Cause_{IV}$ = 0	$SRSCtl_{ESS}$	Treat as exception
Vectored EIC Interrupt	$Cause_{IV}$ = 1 and $Config3_{VEIC}$ = 1	$SRSCtl_{EICSS}$	Source is external interrupt controller.

Register 2-7: SRSCtl: Register; CP0 Register 12, Select 2

r-x	r-x	R-0	R-0	R-0	R-1	r-x	r-x
—	—	HSS<29:26>				—	—
bit 31							bit 24

r-x	r-x	R-x	R-x	R-x	R-x	r-x	r-x
—	—	EICSS<21:18>				—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x	R/W-0	R/W-0
ESS<15:12>				—	—	PSS<9:8>	
bit 15							bit 8

R/W-0	R/W-0	r-0	r-0	R-0	R-0	R-0	R-0
PSS<7:6>		0<5:4>		CSS<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-30 **Reserved:** Write ‘0’; ignore read
Must be written as zeros; returns ‘0’ on read.

bit 29-26 **HSS<29:26>:** High Shadow Set bit
This field contains the highest shadow set number that is implemented by this processor. A value of ‘0’ in this field indicates that only the normal GPRs are implemented.
Possible values of this field for the PIC32MX processor are:

0 = One shadow set (normal GPR set) is present
1 = Two shadow sets are present
3 = Four shadow sets are present
2, 3-15 = Reserved

The value in this field also represents the highest value that can be written to the ESS, EICSS, PSS, and CSS fields of this register, or to any of the fields of the SRSMAP register. The operation of the processor is *undefined* if a value larger than the one in this field is written to any of these other fields.

bit 25-22 **Reserved:** Write ‘0’; ignore read
Must be written as ‘0’; returns ‘0’ on read.

© 2008 Microchip Technology Inc.

Register 2-7: SRSCtl: Register; CP0 Register 12, Select 2 (Continued)

bit 21-18 **EICSS<21:18>:** External Interrupt Controller Shadow Set bits

EIC Interrupt mode shadow set. This field is loaded from the external interrupt controller for each interrupt request and is used in place of the SRSMAP register to select the current shadow set for the interrupt.

bit 17-16 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

bit 15-12 **ESS<15:12>:** Exception Shadow Set bits

This field specifies the shadow set to use on entry to Kernel mode caused by any exception other than a vectored interrupt.
The operation of the processor is *undefined* if software writes a value into this field that is greater than the value in the HSS field.

bit 11-10 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

bit 9-6 **PSS<9:6>:** Previous Shadow Set bits

Since GPR shadow registers are implemented, this field is copied from the CSS field when an exception or interrupt occurs. An `ERET` instruction copies this value back into the CSS field if $Status_{BEV}$ = 0.
This field is not updated on any exception which sets $Status_{ERL}$ to 1 (i.e., Reset, Soft Reset, NMI, cache error), an entry into EJTAG DEBUG mode, or any exception or interrupt that occurs with $Status_{EXL}$ = 1, or $Status_{BEV}$ = 1. This field is not updated on an exception that occurs while $Status_{ERL}$ = 1.
The operation of the processor is *undefined* if software writes a value into this field that is greater than the value in the HSS field.

bit 5-4 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

3-0 **CSS<3:0>:** Current Shadow Set bits

Since GPR shadow registers are implemented, this field is the number of the current GPR set. This field is updated with a new value on any interrupt or exception, and restored from the PSS field on an ERET. Table 2-10 describes the various sources from which the CSS field is updated on an exception or interrupt.
This field is not updated on any exception which sets $Status_{ERL}$ to 1 (i.e., Reset, Soft Reset, NMI, cache error), an entry into EJTAG DEBUG mode, or any exception or interrupt that occurs with $Status_{EXL}$ = 1, or $Status_{BEV}$ = 1. Neither is it updated on an ERET with $Status_{ERL}$ = 1 or $Status_{BEV}$ = 1.
This field is not updated on an exception that occurs while $Status_{ERL}$ = 1.
The value of CSS can be changed directly by software only by writing the PSS field and executing an `ERET` instruction.

© 2008 Microchip Technology Inc.

2.12.8 SRSMAP: Register (CP0 Register 12, Select 3)

The SRSMAP register contains eight 4-bit fields that provide the mapping from an vector number to the shadow set number to use when servicing such an interrupt. The values from this register are not used for a non-interrupt exception, or a non-vectored interrupt ($Cause_{IV}$ = `0` or $IntCtl_{VS}$ = `0`). In such cases, the shadow set number comes from $SRSCtl_{ESS}$.

If $SRSCtl_{HSS}$ is ‘`0`’, the results of a software read or write of this register are *unpredictable*.

The operation of the processor is *undefined* if a value is written to any field in this register that is greater than the value of $SRSCtl_{HSS}$.

The SRSMAP register contains the shadow register set numbers for vector numbers 7..0. The same shadow set number can be established for multiple interrupt vectors, creating a many-to-one mapping from a vector to a single shadow register set number.

Register 2-8: SRSMAP: Register; CP0 Register 12, Select 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV7<31:28>				SSV6<27:24>			
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV5<23:20>				SSV4<19:16>			
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV3<15:12>				SSV2<11:8>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV1<7:4>				SSV0<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-28	**SSV7<31:28>:** Shadow Set Vector 7 bits Shadow register set number for Vector Number 7
bit 27-24	**SSV6<27:24>:** Shadow Set Vector 6 bits Shadow register set number for Vector Number 6
bit 23-20	**SSV5<23:20>:** Shadow Set Vector 5 bits Shadow register set number for Vector Number 5
bit 19-16	**SSV4<19:16>:** Shadow Set Vector 4 bits Shadow register set number for Vector Number 4
bit 15-12	**SSV3<15:12>:** Shadow Set Vector 3 bits Shadow register set number for Vector Number 3
bit 11-8	**SSV2<11:8>:** Shadow Set Vector 2 bits Shadow register set number for Vector Number 2
bit 7-4	**SSV1<7:4>:** Shadow Set Vector 1 bits Shadow register set number for Vector Number 1

 © 2008 Microchip Technology Inc.

Register 2-8: SRSMAP: Register; CP0 Register 12, Select 3 (Continued)

bit 3-0	**SSV0<3:0>:** Shadow Set Vector 0 bit Shadow register set number for Vector Number 0

2.12.9 CAUSE Register (CP0 Register 13, Select 0)

The CAUSE register primarily describes the cause of the most recent exception. In addition, fields also control software interrupt requests and the vector through which interrupts are dispatched. With the exception of the $IP_{1..0}$, DC, IV and WP fields, all fields in the CAUSE register are read-only. $IP_{7..2}$ are interpreted as the Requested Interrupt Priority Level (RIPL).

Table 2-11: Cause Register ExcCode Field

Exception Code Value		Mnemonic	Description
Decimal	Hex		
0	16#00	Int	Interrupt
4	16#04	AdEL	Address error exception (load or instruction fetch)
5	16#05	AdES	Address error exception (store)
6	16#06	IBE	Bus error exception (instruction fetch)
7	16#07	DBE	Bus error exception (data reference: load or store)
8	16#08	Sys	Syscall exception
9	16#09	Bp	Breakpoint exception
10	16#0a	RI	Reserved instruction exception
11	16#0b	CPU	Coprocessor Unusable exception
12	16#0c	Ov	Arithmetic Overflow exception
13	16#0d	Tr	Trap exception
14-18	16#0e-16#12	–	Reserved

 © 2008 Microchip Technology Inc.

Register 2-9: CAUSE: Register; CP0 Register 13, Select 0

R-x	R-x	R-x	R-x	R/W-0	R-0	r-x	r-x
BD	TI	CE<29:28>		DC	R	0<25:24>	
bit 31							bit 24

R/W-x	R/W-0	r-x	r-x	r-x	r-x	r-x	r-x
IV	R	0<21:16>					
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R/W-x	R/W-x
RIPL<15:10>						IP1..IP0<9:8>	
bit 15							bit 8

r-x	R-x	R-x	R-x	R-x	R-x	r-x	r-x
0	EXCCODE<6:2>					0<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31 **BD:** Branch Delay bit

Indicates whether the last exception taken occurred in a branch delay slot:
0 = Not in delay slot
1 = In delay slot
The processor updates BD only if $Status_{EXL}$ was '0' when the exception occurred.

bit 30 **TI:** Timer Interrupt bit

Timer Interrupt. This bit denotes whether a timer interrupt is pending (analogous to the IP bits for other interrupt types):
0 = No timer interrupt is pending
1 = Timer interrupt is pending

bit 29-28 **CE<29:28>:** Coprocessor Exception bits

Coprocessor unit number referenced when a Coprocessor Unusable exception is taken. This field is loaded by hardware on every exception, but is *unpredictable* for all exceptions except for Coprocessor Unusable.

bit 27 **DC:** Disable Count bit

Disable Count register. In some power-sensitive applications, the COUNT register is not used and can be stopped to avoid unnecessary toggling
0 = Enable counting of COUNT register
1 = Disable counting of COUNT register

bit 26 **R:** bit

bit 25-24 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

Register 2-9: CAUSE: Register; CP0 Register 13, Select 0 (Continued)

bit 23	**IV:** Interrupt Vector bit Indicates whether an interrupt exception uses the general exception vector or a special interrupt vector 0 = Use the general exception vector (16#180) 1 = Use the special interrupt vector (16#200) If the $Cause_{IV}$ is 1 and $Status_{BEV}$ is 0, the special interrupt vector represents the base of the vectored interrupt table.
bit 22	**R:** bit
bit 21-16	**Reserved:** Write ‘0’; ignore read Must be written as ‘0’; returns ‘0’ on read.
bit 15-10	**RIPL<15:10>:** Requested Interrupt Priority Level bits Requested Interrupt Priority Level.\ This field is the encoded (0..63) value of the requested interrupt. A value of ‘0’ indicates that no interrupt is requested.
bit 9-8	**IP1..IP0<9:8>:** Controls the request for software interrupts: 0 = No interrupt requested 1 = Request software interrupt These bits are exported to the system interrupt controller for prioritization in EIC interrupt mode with other interrupt sources
bit 7	**Reserved:** Write ‘0’; ignore read Must be written as ‘0’; returns ‘0’ on read.
bit 6-2	**EXCCODE<6:2>:** Exception Code bits Exception code - see Table 2-11
bit 1-0	**Reserved:** Write ‘0’; ignore read Must be written as ‘0’; returns ‘0’ on read.

 © 2008 Microchip Technology Inc.

2.12.10 EPC Register (CP0 Register 14, Select 0)

The Exception Program Counter (EPC) is a read/write register that contains the address at which processing resumes after an exception has been serviced. All bits of the EPC register are significant and are writable.

For synchronous (precise) exceptions, the EPC contains one of the following:

- The virtual address of the instruction that was the direct cause of the exception.
- The virtual address of the immediately preceding `BRANCH` or `JUMP` instruction, when the exception causing instruction is in a branch delay slot and the Branch Delay bit in the CAUSE register is set.

On new exceptions, the processor does not write to the EPC register when the EXL bit in the STATUS register is set, however, the register can still be written via the `MTC0` instruction.

Since the PIC32 family implements MIPS16e ASE, a read of the EPC register (via MFC0) returns the following value in the destination GPR:

$$\mathtt{GPR[rt]} \leftarrow \mathtt{ExceptionPC}_{31..1} \;||\; \mathtt{ISAMode}_0$$

That is, the upper 31 bits of the exception PC are combined with the lower bit of the ISAMode field and written to the GPR.

Similarly, a write to the EPC register (via MTC0) takes the value from the GPR and distributes that value to the exception PC and the ISAMode field, as follows

$$\mathtt{ExceptionPC} \leftarrow \mathtt{GPR[rt]}_{31..1} \;||\; 0$$
$$\mathtt{ISAMode} \leftarrow 2\#0 \;||\; \mathtt{GPR[rt]}_0$$

That is, the upper 31 bits of the GPR are written to the upper 31 bits of the exception PC, and the lower bit of the exception PC is cleared. The upper bit of the ISAMode field is cleared and the lower bit is loaded from the lower bit of the GPR.

Register 2-10: EPC: Register; CP0 Register 14, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **EPC<31:0>:** Exception Program Counter bits

2.12.11 PRID Register (CP0 Register 15, Select 0)

The Processor Identification (PRID) register is a 32 bit read-only register that contains information identifying the manufacturer, manufacturer options, processor identification, and revision level of the processor.

Register 2-11: PRID: Register; CP0 Register 15, Select 0

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
R<31:24>							
bit 31							bit 24

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
COMPANY ID<23:16>							
bit 23							bit 16

R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87
PROCESSOR ID<15:8>							
bit 15							bit 8

R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset
REVISION<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-24 **R<31:24>:** Reserved
Must be ignored on write and read as ‘0’

bit 23-16 **COMPANY ID<23:16>:**
Identifies the company that designed or manufactured the processor. In the PIC32MX this field contains a value of 1 to indicate MIPS Technologies, Inc.

bit 15-8 **PROCESSOR ID<15:8>:**
Identifies the type of processor. This field allows software to distinguish between the various types of MIPS Technologies processors.

bit 7-0 **REVISION<7:0>:**
Specifies the revision number of the processor. This field allows software to distinguish between one revision and another of the same processor type.
This field is broken up into the following three subfields.

bit 7-5 **MAJOR REVISION<7:5>:**
This number is increased on major revisions of the processor core.

bit 4-2 **MINOR REVISION<4:2>:**
This number is increased on each incremental revision of the processor and reset on each new major revision.

bit 1-0 **PATCH LEVEL<1:0>:**
If a patch is made to modify an older revision of the processor, this field will be incremented.

 © 2008 Microchip Technology Inc.

2.12.12 EBASE Register (CP0 Register 15, Select 1)

The EBASE register is a read/write register containing the base address of the exception vectors used when $Status_{BEV}$ equals ‘0’, and a read-only CPU number value that may be used by software to distinguish different processors in a multi-processor system.

The EBASE register provides the ability for software to identify the specific processor within a multi-processor system, and allows the exception vectors for each processor to be different, especially in systems composed of heterogeneous processors. Bits 31..12 of the EBASE register are concatenated with zeros to form the base of the exception vectors when $Status_{BEV}$ is ‘0’. The exception vector base address comes from the fixed defaults when $Status_{BEV}$ is ‘1’, or for any EJTAG Debug exception. The Reset state of bits 31..12 of the EBASE register initialize the exception base register to 16#8000.0000.

Bits 31..30 of the EBASE Register are fixed with the value 2#10 to force the exception base address to be in the kseg0 or kseg1 unmapped virtual address segments.

If the value of the exception base register is to be changed, this must be done with $Status_{BEV}$ equal ‘1’. The operation of the processor is *undefined* if the Exception Base field is written with a different value when $Status_{BEV}$ is ‘0’.

Combining bits 31..20 with the Exception Base field allows the base address of the exception vectors to be placed at any 4 KBbyte page boundary.

Register 2-12: EBASE: Register; CP0 Register 15, Select 1

R-1	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
1	0	EXCEPTION BASE<29:24>					
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EXCEPTION BASE<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-0	r-0	R-0	R-0
EXCEPTION BASE<15:12>				r		CPUNUM<9:8>	
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CPUNUM<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **1:** One bit
This bit is ignored on write and returns one on read.

bit 30 **0:** Zero bit
This bit is ignored on write and returns ‘0’ on read.

bit 29-12 **EXCEPTION BASE<29:12>:**
In conjunction with bits 31..30, this field specifies the base address of the exception vectors when $Status_{BEV}$ is ‘0’.

bit 11-10 **Reserved:**
Must be written as ‘0’; returns ‘0’ on read.

Register 2-12: EBASE: Register; CP0 Register 15, Select 1 (Continued)

bit 9-0 **CPUNUM<9:0>:**

This field specifies the number of the CPU in a multi-processor system and can be used by software to distinguish a particular processor from the others. In a single processor system, this value is set to ‘0’.

 © 2008 Microchip Technology Inc.

2.12.13 CONFIG Register (CP0 Register 16, Select 0)

The CONFIG register specifies various configuration and capabilities information. Most of the fields in the CONFIG register are initialized by hardware during the Reset exception process, or are constant.

Table 2-12: Cache Coherency Attributes

C(2:0) Value	Cache Coherency Attribute
2	Uncached
3	Cacheable

Register 2-13: CONFIG: Register; CP0 Register 16, Select 0

R-1	R-0	R-1	R-0	R/W-0	R/W-1	R/W-0	r-0
M	K23<30:28>			KU<27:25>			0
bit 31							bit 24

r-x	R-0	R-0	R-0	r-x	r-x	r-x	R-1
0	UDI	SB	MDU				DS
bit 23							bit 16

R-0	R-0	R-0	R-0	R-0	R-1	R-0	R-1
BE	AT<14:13>		AR<12:10>			MT<9:8>	
bit 15							bit 8

R-1	r-x	r-x	r-x	r-x	R/W-0	R/W-1	R/W-0
MT					K0<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:**
This bit is hardwired to ‘1’ to indicate the presence of the CONFIG1 register.

bit 30-28 **K23<30:28>:** kseg2 and kseg3 bits
This field controls the cacheability of the kseg2 and kseg3 address segments.
Refer to Table 2-12 for the field encoding.

bit 27-25 **KU<27:25>:** kuseg and useg bits
This field controls the cacheability of the kuseg and useg address segments.
Refer to Table 2-12 for the field encoding.

bit 24-23 **Reserved**: Write ‘0’; ignore read
Must be written as ‘0’. Returns ‘0’ on reads.

bit 22 **UDI:** User Defined bit
This bit indicates that CorExtend User Defined Instructions have been implemented.
0 = No User Defined Instructions are implemented
1 = User Defined Instructions are implemented

Register 2-13: CONFIG: Register; CP0 Register 16, Select 0 (Continued)

bit 21	**SB:** SimpleBE bit Indicates whether SimpleBE Bus mode is enabled. 0 = No reserved byte enables on internal bus interface 1 = Only simple byte enables allowed on internal bus interface
bit 20	**MDU:** Multiply/Divide Unit bit This bit indicates the type of Multiply/Divide Unit present 0 = Fast, high-performance MDU
bit 19-17	**Reserved:** Write '0'; ignore read Must be written as 0. Returns '0' on reads.
bit 16	**DS:** Dual SRAM bit 0 = Unified instruction/data SRAM internal bus interface 1 = Dual instruction/data SRAM internal bus interfaces **Note:** The PIC32MX family currently uses Dual SRAM-style interfaces internally.
bit 15	**BE:** Big Endian bit Indicates the Endian mode in which the processor is running, PIC32MX is always little endian. 0 = Little endian 1 = Big enidan
bit 14-13	**AT<14:13>:** Architecture Type bits Architecture type implemented by the processor. This field is always '00' to indicate the MIPS32 architecture.
bit 12-10	**AR<12:10>:** Architecture Revision Level bits Architecture revision level. This field is always '001' to indicate MIPS32 Release 2. 0: Release 1 1: Release 2 2-7: Reserved
bit 9-7	**MT<9:7>:** MMU Type bits 3: Fixed mapping 0-2, 4-7: Reserved
bit 6-3	**Reserved:** Write '0'; ignore read Must be written as zeros; returns zeros on reads
bit 2-0	**K0<2:0>:** Kseg0 bits Kseg0 coherency algorithm. Refer to XREF Table 2-12 for the field encoding.

 © 2008 Microchip Technology Inc.

2.12.14 CONFIG1 Register (CP0 Register 16, Select 1)

The CONFIG1 register is an adjunct to the CONFIG register and encodes additional information about capabilities present on the core. All fields in the CONFIG1 register are read-only.

Register 2-14: CONFIG1: CONFIG1 Register; CP0 Register 16, Select 1

R-1	R-x	R-x	R-x	R-x	R-x	R-x	R-x
M	MMU Size<30:25>						IS
bit 31							bit 24

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
IS<23:22>		IL<21:19>			IA<18:16>		
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
DS<15:13>			DL<12:10>			DA<9:8>	
bit 15							bit 8

R-x	R-0	R-0	R-0	R-0	R-1	R-x	R-0
DA	C2	MD	PC	WR	CA	EP	FP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:** bit
This bit is hardwired to ‘1’ to indicate the presence of the CONFIG2 register.

bit 30-25 **MMU Size:** bits
This field contains the number of entries in the TLB minus one; since the PIC32MX has no TLB, this field is ‘0’.

bit 24-22 **IS:** Instruction Cache Sets bits
This field contains the number of instruction cache sets per way; since the M4K core does not include caches, this field is always read as ‘0’.

bit 21-19 **IL:** Instruction-Cache Line bits
This field contains the instruction cache line size; since the M4K core does not include caches, this field is always read as ‘0’.

bit 18-16 **IA:** Instruction-Cache Associativity bits
This field contains the level of instruction cache associativity; since the M4K core does not include caches, this field is always read as ‘0’.

bit 15-13 **DS:** Data-Cache Sets bits
This field contains the number of data cache sets per way; since the M4K core does not include caches, this field is always read as ‘0’.

bit 12-10 **DL:** Data-Cache Line bits
This field contains the data cache line size; since the M4K core does not include caches, this field is always read as ‘0’.

bit 9-7 **DA:** Data-Cache Associativity bits
This field contains the type of set associativity for the data cache; since the M4K core does not include caches, this field is always read as ‘0’.

Register 2-14: CONFIG1: CONFIG1 Register; CP0 Register 16, Select 1 (Continued)

bit 6	C2: Coprocessor 2 bit Coprocessor 2 present. `0` = No coprocessor is attached to the COP2 interface `1` = A coprocessor is attached to the COP2 interface Since coprocessor 2 is not implemented in the PIC32MX family of microcontrollers, this bit will read '`0`'.
bit 5	**MD:** MDMX bit MDMX implemented. This bit always reads as '`0`' because MDMX is not supported.
bit 4	**PC:** Performance Counter bit Performance Counter registers implemented. Always a '`0`' since the PIC32MX core does not contain Performance Counters.
bit 3	**WR:** Watch Register bit Watch registers implemented. `0` = No Watch registers are present `1` = One or more Watch registers are present **Note:** The PIC32MX does not implement watch registers, therefore this bit always reads '`0`'.
bit 2	**CA:** Code Compression Implemented bit `0` = No MIPS16e present `1` = MIPS16e is implemented
bit 1	**EP:** EJTAG Present bit This bit is always set to indicate that the core implements EJTAG.
bit 0	**FP:** FPU Implemented bit This bit is always '`0`' since the core does not contain a floating point unit.

 © 2008 Microchip Technology Inc.

2.12.15 CONFIG2 (CP0 Register 16, Select 2)

The CONFIG2 register is an adjunct to the CONFIG register and is reserved to encode additional capabilities information. CONFIG2 is allocated for showing the configuration of level 2/3 caches. These fields are reset to ‘0’ because L2/L3 caches are not supported by the PIC32MX core. All fields in the CONFIG2 register are read-only.

Register 2-15: CONFIG2: CONFIG2 Register; CP0 Register 16, Select 2

R-1	r-0	r-0	r-0	r-0	r-0	r-0	r-0
M	0	0	0	0	0	0	0
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 15							bit 8

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:** bit
This bit is hardwired to ‘1’ to indicate the presence of the CONFIG3 register.

bit 30-0 **Reserved**

2.12.16 CONFIG3 Register (CP0 Register 16, Select 3)

The CONFIG3 register encodes additional capabilities. All fields in the CONFIG3 register are read-only.

Register 2-16: CONFIG3: CONFIG3 Register; CP0 Register 16, Select 3

R-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
M	0	0	0	0	0	0	0
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 15							bit 8

r-0	R-1	R-1	R-0	r-0	r-0	R-0	R-0
0	VEIC	VInt	SP	0	0	SM	TL
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:** Reserved
This bit is reserved to indicate that a CONFIG4 register is present. With the current architectural definition, this bit should always read as a ‘0’.

bit 30-7 **Reserved:** Write ‘0’; ignore read
Must be written as zeros; returns zeros on read.

bit 6 **VEIC:**
Support for an external interrupt controller is implemented.
0 = Support for EIC Interrupt mode is not implemented
1 = Support for EIC Interrupt mode is implemented

Note: PIC32MX internally implements a MIPS “external interrupt controller”, therefore this bit reads ‘1’.

bit 5 **VINT:** Vector Interrupt bit
Vectored interrupts implemented. This bit indicates whether vectored interrupts are implemented.
0 = Vector interrupts are not implemented
1 = Vector interrupts are implemented
On the PIC32MX core, this bit is always a ‘1’ since vectored interrupts are implemented.

bit 4 **SP:** Support Page bit
Small (1 KByte) page support is implemented, and the PAGEGRAIN register exists.
0 = Small page support is not implemented
1 = Small page support is implemented

Note: PIC32MX always reads ‘0’ since PIC32MX does not implement small page support.

bit 3-2 **0:**
Must be written as zeros; returns zeros on read.

 © 2008 Microchip Technology Inc.

Register 2-16: CONFIG3: CONFIG3 Register; CP0 Register 16, Select 3 (Continued)

bit 1 **SM:** SmartMIPS™ bit

SmartMIPS™ ASE implemented. This bit indicates whether the SmartMIPS ASE is implemented. Since SmartMIPS is present on the PIC32MX core, this bit will always be '0'.
0 = SmartMIPS ASE is not implemented
1 = SmartMIPS ASE is implemented

bit 0 **TL:** Trace Logic bit

Trace Logic implemented. This bit indicates whether PC or data trace is implemented.
0 = On-chip trace logic (PDTrace™) is not implemented
1 = On-chip trace logic (PDTrace™) is implemented

Note: PIC32MX does not implement PDTrace™ on-chip trace logic, therefore this bit always reads '0'.

2.12.17 DEBUG Register (CP0 Register 23, Select 0)

The DEBUG register is used to control the debug exception and provide information about the cause of the debug exception and when re-entering at the debug exception vector due to a normal exception in DEBUG mode. The read-only information bits are updated every time the debug exception is taken or when a normal exception is taken when already in DEBUG mode.

Only the DM bit and the EJTAGver field are valid when read from Non-DEBUG mode; the values of all other bits and fields are *unpredictable*. Operation of the processor is *undefined* if the DEBUG register is written from Non-DEBUG mode.

Some of the bits and fields are only updated on debug exceptions and/or exceptions in DEBUG mode, as shown below:

- DSS, DBp, DDBL, DDBS, DIB, DINT are updated on both debug exceptions and on exceptions in Debug modes
- DExcCode is updated on exceptions in DEBUG mode, and is undefined after a debug exception
- Halt and Doze are updated on a debug exception, and are undefined after an exception in DEBUG mode
- DBD is updated on both debug and on exceptions in Debug modes

All bits and fields are undefined when read from normal mode, except EJTAGver and DM.

 Preliminary © 2008 Microchip Technology Inc.

Register 2-17: DEBUG: Register; CP0 Register 23, Select 0

R-U	R-0	R-0	R/W-0	R-U	R-U	R/W-1	R/W-0
DBD	DM	NODCR	LSNM	DOZE	HALT	COUNTDM	IBUSEP
bit 31							bit 24

R-0	R-0	R/W-0	R/W-0	R-0	R-0	R-0	R-1
MCHECKP	CACHEEP	DBUSEP	IEXI	DDBSIMPR	DDBLIMPR	VER<7:6>	
bit 23							bit 16

R-0	R-U	R-U	R-U	R-U	R-U	R-0	R/W-0
VER	DEXCCODE<14:10					NOSST	SST
bit 15							bit 8

R-0	R-0	R-U	R-U	R-U	R-U	R-U	R-U
R<7:6>		DINT	DIB	DDBS	DDBL	DBP	DSS
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **DBD:**
Indicates whether the last debug exception or exception in DEBUG mode, occurred in a branch delay slot:
0 = Not in delay slot
1 = In delay slot

bit 30 **DM:**
Indicates that the processor is operating in DEBUG mode:
0 = Processor is operating in Non-DEBUG mode
1 = Processor is operating in DEBUG mode

bit 29 **NODCR:**
Indicates whether the dseg memory segment is present and the Debug Control Register is accessible:
0 = dseg is present
1 = No dseg present

bit 28 **LSNM:**
Controls access of load/store between dseg and main memory:
0 = Load/stores in dseg address range goes to dseg
1 = Load/stores in dseg address range goes to main memory

bit 27 **DOZE:**
Indicates that the processor was in any kind of Low-Power mode when a debug exception occurred:
0 = Processor not in Low-Power mode when debug exception occurred
1 = Processor in Low-Power mode when debug exception occurred

bit 26 **HALT:**
Indicates that the internal system bus clock was stopped when the debug exception occurred:
0 = Internal system bus clock stopped
1 = Internal system bus clock running

Register 2-17: DEBUG: Register; CP0 Register 23, Select 0 (Continued)

bit 25 **COUNTDM:**
Indicates the Count register behavior in DEBUG mode.
0 = Count register stopped in DEBUG mode
1 = Count register is running in DEBUG mode

bit 24 **IBUSEP:**
Instruction fetch Bus Error exception Pending. Set when an instruction fetch bus error event occurs or if a '1' is written to the bit by software. Cleared when a Bus Error exception on instruction fetch is taken by the processor, and by Reset. If IBUSEP is set when IEXI is cleared, a Bus Error exception on instruction fetch is taken by the processor, and IBUSEP is cleared.

bit 23 **MCHECKP:**
Indicates that an imprecise Machine Check exception is pending. All Machine Check exceptions are precise on the PIC32MX processor so this bit will always read as '0'.

bit 22 **CACHEEP:**
Indicates that an imprecise Cache Error is pending. Cache Errors cannot be taken by the PIC32MX core so this bit will always read as '0'.

bit 21 **DBUSEP:**
Data access Bus Error exception Pending. Covers imprecise bus errors on data access, similar to behavior of IBUSEP for imprecise bus errors on an instruction fetch.

bit 20 **IEXI:**
Imprecise Error eXception Inhibit controls exceptions taken due to imprecise error indications. Set when the processor takes a debug exception or exception in DEBUG mode. Cleared by execution of the DERET instruction; otherwise modifiable by DEBUG mode software. When IEXI is set, the imprecise error exception from a bus error on an instruction fetch or data access, cache error, or machine check is inhibited and deferred until the bit is cleared.

bit 19 **DDBSIMPR:**
Indicates that an imprecise Debug Data Break Store exception was taken. All data breaks are precise on the PIC32MX core, so this bit will always read as '0'.

bit 18 **DDBLIMPR:**
Indicates that an imprecise Debug Data Break Load exception was taken. All data breaks are precise on the PIC32MX core, so this bit will always read as '0'.

bit 17-15 **VER:**
EJTAG version

bit 14-10 **DEXCCODE:**
Indicates the cause of the latest exception in DEBUG mode. The field is encoded as the ExcCode field in the CAUSE register for those normal exceptions that may occur in DEBUG mode.
Value is undefined after a debug exception.

bit 9 **NOSST:**
Indicates whether the single-step feature controllable by the SST bit is available in this implementation:
0 = Single-step feature available
1 = No single-step feature available

bit 8 **SST:**
Controls if debug single step exception is enabled:
0 = No debug single-step exception enabled
1 = Debug single step exception enabled

bit 7-6 **Reserved:**
Must be written as zeros; returns zeros on reads.

bit 5 **DINT:**
Indicates that a debug interrupt exception occurred. Cleared on exception in DEBUG mode.
0 = No debug interrupt exception
1 = Debug interrupt exception

 © 2008 Microchip Technology Inc.

Register 2-17: DEBUG: Register; CP0 Register 23, Select 0 (Continued)

bit 4 **DIB:**
Indicates that a debug instruction break exception occurred. Cleared on exception in DEBUG mode.
0 = No debug instruction exception
1 = Debug instruction exception

bit 3 **DDBS:**
Indicates that a debug data break exception occurred on a store. Cleared on exception in DEBUG mode.
0 = No debug data exception on a store
1 = Debug instruction exception on a store

bit 2 **DDBL:**
Indicates that a debug data break exception occurred on a load. Cleared on exception in DEBUG mode.
0 = No debug data exception on a load
1 = Debug instruction exception on a load

bit 1 **DBP:**
Indicates that a debug software breakpoint exception occurred. Cleared on exception in DEBUG mode.
0 = No debug software breakpoint exception
1 = Debug software breakpoint exception

bit 0 **DSS:**
Indicates that a debug single-step exception occurred. Cleared on exception in DEBUG mode.
0 = No debug single-step exception
1 = Debug single-step exception

2.12.18 DEPC Register (CP0 Register 24, Select 0)

The Debug Exception Program Counter (DEPC) register is a read/write register that contains the address at which processing resumes after a debug exception or DEBUG mode exception has been serviced.

For synchronous (precise) debug and DEBUG mode exceptions, the DEPC contains either:

- The virtual address of the instruction that was the direct cause of the debug exception, or
- The virtual address of the immediately preceding branch or jump instruction, when the debug exception causing instruction is in a branch delay slot, and the Debug Branch Delay (DBD) bit in the Debug register is set.

For asynchronous debug exceptions (debug interrupt), the DEPC contains the virtual address of the instruction where execution should resume after the debug handler code is executed.

Since the PIC32 family implements the MIPS16e ASE, a read of the DEPC register (via MFC0) returns the following value in the destination GPR:

$$GPR[rt] = DebugExceptionPC_{31..1} \,||\, ISAMode_0$$

That is, the upper 31 bits of the debug exception PC are combined with the lower bit of the ISAMode field and written to the GPR.

Similarly, a write to the DEPC register (via MTC0) takes the value from the GPR and distributes that value to the debug exception PC and the ISAMode field, as follows:

$$DebugExceptionPC = GPR[rt]_{31..1} \,||\, 0$$
$$ISAMode = 2\#0 \,||\, GPR[rt]_0$$

That is, the upper 31 bits of the GPR are written to the upper 31 bits of the debug exception PC, and the lower bit of the debug exception PC is cleared. The upper bit of the ISAMode field is cleared and the lower bit is loaded from the lower bit of the GPR.

 © 2008 Microchip Technology Inc.

Register 2-18: DEPC: Debug Exception Program Counter Register; CP0 Register 24, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DEPC<31:0>:** Debug Exception Program Counter bits

The DEPC register is updated with the virtual address of the instruction that caused the debug exception. If the instruction is in the branch delay slot, then the virtual address of the immediately preceding branch or jump instruction is placed in this register.

Execution of the `DERET` instruction causes a jump to the address in the DEPC.

2.12.19 ErrorEPC (CP0 Register 30, Select 0)

The ErrorEPC register is a read/write register, similar to the EPC register, except that ErrorEPC is used on error exceptions. All bits of the ErrorEPC register are significant and must be writable. It is also used to store the program counter on Reset, Soft Reset, and nonmaskable interrupt (NMI) exceptions.

The ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error. This address can be:

- The virtual address of the instruction that caused the exception
- The virtual address of the immediately preceding branch or jump instruction when the error causing instruction is in a branch delay slot

Unlike the EPC register, there is no corresponding branch delay slot indication for the ErrorEPC register.

Since the PIC32 family implements the MIPS16e ASE, a read of the ErrorEPC register (via MFC0) returns the following value in the destination GPR:

$$GPR[rt] = ErrorExceptionPC_{31..1} \mid\mid ISAMode_0$$

That is, the upper 31 bits of the error exception PC are combined with the lower bit of the ISAMode field and written to the GPR.

Similarly, a write to the ErrorEPC register (via MTC0) takes the value from the GPR and distributes that value to the error exception PC and the ISAMode field, as follows:

$$ErrprExceptionPC = GPR[rt]_{31..1} \mid\mid 0$$
$$ISAMode = 2\#0 \mid\mid GPR[rt]_0$$

That is, the upper 31 bits of the GPR are written to the upper 31 bits of the error exception PC, and the lower bit of the error exception PC is cleared. The upper bit of the ISAMode field is cleared and the lower bit is loaded from the lower bit of the GPR.

Register 2-19: ErrorEPC: Error Exception Program Counter Register; CP0 Register 30, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **ErrorEPC<31:0>:** Error Exception Program Counter bits

 © 2008 Microchip Technology Inc.

2.12.20 DeSave Register (CP0 Register 31, Select 0)

The Debug Exception Save (DeSave) register is a read/write register that functions as a simple memory location. This register is used by the debug exception handler to save one of the GPRs that is then used to save the rest of the context to a pre-determined memory area (such as in the EJTAG Probe). This register allows the safe debugging of exception handlers and other types of code where the existence of a valid stack for context saving cannot be assumed.

Register 2-20: DeSave: Debug Exception Save Register; CP0 Register 31, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DESAVE<31:0>:** Debug Exception Save bits
Scratch Pad register used by Debug Exception code.

© 2008 Microchip Technology Inc.

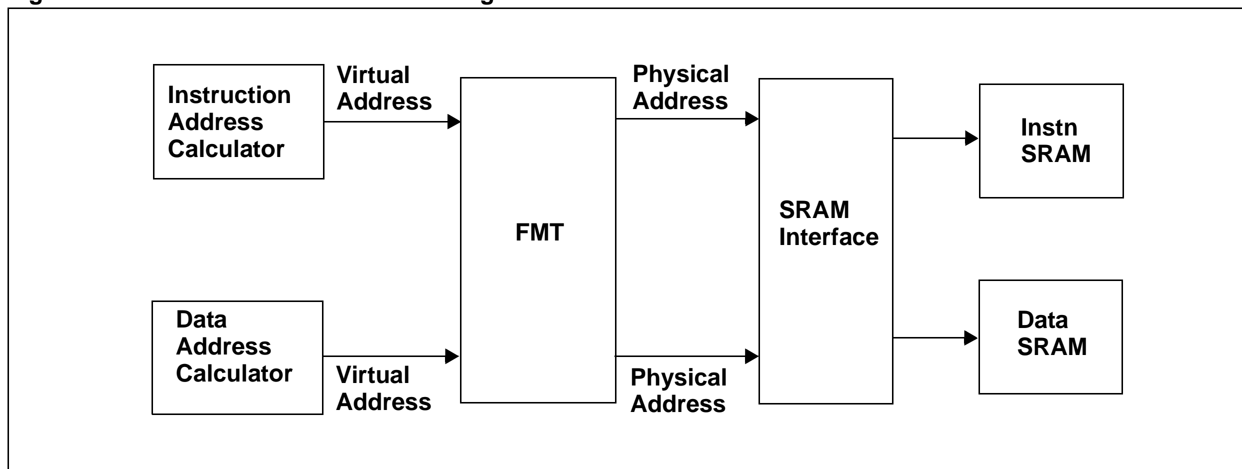
2.13 MIPS16e™ EXECUTION

When the core is operating in MIPS16e mode, instruction fetches only require 16-bits of data to be returned. For improved efficiency, however, the core will fetch 32-bits of instruction data whenever the address is word-aligned. Thus for sequential MIPS16e code, fetches only occur for every other instruction, resulting in better performance and reduced system power.

2.14 MEMORY MODEL

Virtual addresses used by software are converted to physical addresses by the memory management unit (MMU) before being sent to the CPU busses. The PIC32MX CPU uses a fixed mapping for this conversion. For more information regarding the system memory model, see **Section 3. "Memory Organization"**.

Figure 2-13: Address Translation During SRAM Access



2.14.1 Cacheability

The CPU uses the virtual address of an instruction fetch, load or store to determine whether to access the cache or not. Memory accesses within kseg0, or useg/kuseg can be cached, while accesses within kseg1 are non-cacheable. The CPU uses the CCA bits in the CONFIG register to determine the cacheability of a memory segment. A memory access is cacheable if its corresponding CCA = 011_2.

For more information on cache operation, see **Section 4. "Prefetch Cache Module"**.

2.14.1.1 Little Endian Byte Ordering

On CPUs that address memory with byte resolution, there is a convention for multi-byte data items that specify the order of high-order to low-order bytes. Big-endian byte-ordering is where the lowest address has the Most Significant Byte. Little-endian ordering is where the lowest address has the Least Significant Byte of a multi-byte datum. The PIC32MX CPU family supports little-endian byte ordering.

 © 2008 Microchip Technology Inc.

Figure 2-14: Big Endian Byte Ordering

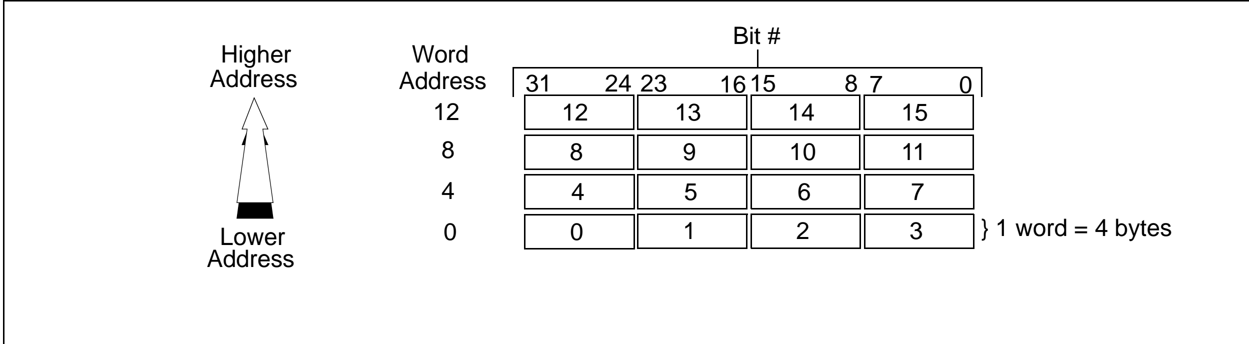
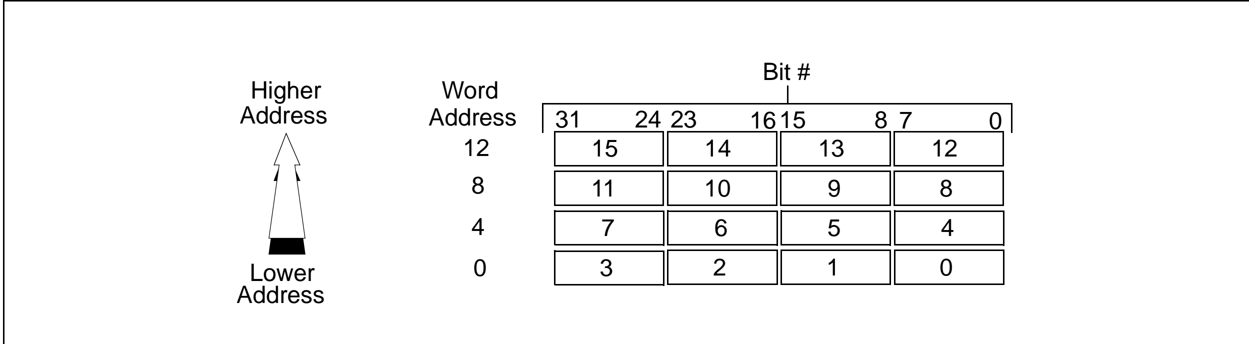


Figure 2-15: Little Endian Byte Ordering



2.15 CPU INSTRUCTIONS, GROUPED BY FUNCTION

CPU instructions are organized into the following functional groups:

- Load and store
- Computational
- Jump and branch
- Miscellaneous
- Coprocessor

Each instruction is 32 bits long.

2.15.1 CPU Load and Store Instructions

MIPS processors use a load/store architecture; all operations are performed on operands held in processor registers and main memory is accessed only through load and store instructions.

2.15.1.1 Types of Loads and Stores

There are several different types of load and store instructions, each designed for a different purpose:

- Transferring variously-sized fields (for example, LB, SW)
- Trading transferred data as signed or unsigned integers (for example, LHU)
- Accessing unaligned fields (for example, LWR, SWL)
- Atomic memory update (read-modify-write: for instance, LL/SC)

2.15.1.2 List of CPU Load and Store Instructions

The following data sizes (as defined in the *AccessLength* field) are transferred by CPU load and store instructions:

- Byte
- Halfword
- Word

Signed and unsigned integers of different sizes are supported by loads that either sign-extend or zero-extend the data loaded into the register.

Unaligned words and doublewords can be loaded or stored in just two instructions by using a pair of special instructions. For loads a LWL instruction is paired with a LWR instruction. The load instructions read the left-side or right-side bytes (left or right side of register) from an aligned word and merge them into the correct bytes of the destination register.

2.15.1.3 Loads and Stores Used for Atomic Updates

The paired instructions, Load Linked and Store Conditional, can be used to perform an atomic read-modify-write of word or doubleword cached memory locations. These instructions are used in carefully coded sequences to provide one of several synchronization primitives, including test-and-set, bit-level locks, semaphores, and sequencers and event counts.

2.15.1.4 Coprocessor Loads and Stores

If a particular coprocessor is not enabled, loads and stores to that processor cannot execute and the attempted load or store causes a Coprocessor Unusable exception. Enabling a coprocessor is a privileged operation provided by the System Control Coprocessor, CP0.

 © 2008 Microchip Technology Inc.

2.15.2 Computational Instructions

Two's complement arithmetic is performed on integers represented in 2s complement notation. These are signed versions of the following operations:

- Add
- Subtract
- Multiply
- Divide

The add and subtract operations labelled "unsigned" are actually modulo arithmetic without overflow detection.

There are also unsigned versions of *multiply* and *divide*, as well as a full complement of *shift* and *logical* operations. Logical operations are not sensitive to the width of the register.

MIPS32 provided 32-bit integers and 32-bit arithmetic.

2.15.2.1 Shift Instructions

The ISA defines two types of shift instructions:

- Those that take a fixed shift amount from a 5-bit field in the instruction word (for instance, SLL, SRL)
- Those that take a shift amount from the low-order bits of a general register (for instance, SRAV, SRLV)

2.15.2.2 Multiply and Divide Instructions

The multiply instruction performs 32-bit by 32-bit multiplication and creates either 64-bit or 32-bit results. Divide instructions divide a 64-bit value by a 32-bit value and create 32-bit results. With one exception, they deliver their results into the HI and LO special registers. The MUL instruction delivers the lower half of the result directly to a GPR.

- Multiply produces a full-width product twice the width of the input operands; the low half is loaded into LO and the high half is loaded into HI.
- Multiply-Add and Multiply-Subtract produce a full-width product twice the width of the input operations and adds or subtracts the product from the concatenated value of HI and LO. The low half of the addition is loaded into LO and the high half is loaded into HI.
- Divide produces a quotient that is loaded into LO and a remainder that is loaded into HI.

The results are accessed by instructions that transfer data between HI/LO and the general registers.

2.15.3 Jump and Branch Instructions

2.15.3.1 Types of Jump and Branch Instructions Defined by the ISA

The architecture defines the following jump and branch instructions:

- PC-relative conditional branch
- PC-region unconditional jump
- Absolute (register) unconditional jump
- A set of procedure calls that record a return link address in a general register.

2.15.3.2 Branch Delays and the Branch Delay Slot

All branches have an architectural delay of one instruction. The instruction immediately following a branch is said to be in the **branch delay slot**. If a branch or jump instruction is placed in the branch delay slot, the operation of both instructions is undefined.

By convention, if an exception or interrupt prevents the completion of an instruction in the branch delay slot, the instruction stream is continued by re-executing the branch instruction. To permit this, branches must be restartable; procedure calls may not use the register in which the return link is stored (usually GPR 31) to determine the branch target address.

2.15.3.3 Branch and Branch Likely

There are two versions of conditional branches; they differ in the manner in which they handle the instruction in the delay slot when the branch is not taken and execution falls through.

- Branch instructions execute the instruction in the delay slot.
- Branch likely instructions do not execute the instruction in the delay slot if the branch is not taken (they are said to nullify the instruction in the delay slot).

  Although the Branch Likely instructions are included in this specification, software is strongly encouraged to avoid the use of the Branch Likely instructions, as they will be removed from a future revision of the MIPS Architecture.

2.15.4 Miscellaneous Instructions

2.15.4.1 Instruction Serialization (SYNC and SYNCI)

In normal operation, the order in which load and store memory accesses appear to a viewer *outside* the executing processor (for instance, in a multiprocessor system) is not specified by the architecture.

The SYNC instruction can be used to create a point in the executing instruction stream at which the relative order of some loads and stores can be determined: loads and stores executed before the SYNC are completed before loads and stores after the SYNC can start.

The SYNCI instruction synchronizes the processor caches with previous writes or other modifications to the instruction stream.

2.15.4.2 Exception Instructions

Exception instructions transfer control to a software exception handler in the kernel. There are two types of exceptions, conditional and unconditional. These are caused by the following instructions: syscall, trap, and break.

Trap instructions, which cause conditional exceptions based upon the result of a comparison

System call and breakpoint instructions, which cause unconditional exceptions

2.15.4.3 Conditional Move Instructions

MIPS32 includes instructions to conditionally move one CPU general register to another, based on the value in a third general register.

 © 2008 Microchip Technology Inc.

2.15.4.4 NOP Instructions

The NOP instruction is actually encoded as an all-zero instruction. MIPS processors special-case this encoding as performing no operation, and optimize execution of the instruction. In addition, SSNOP instruction, takes up one issue cycle on any processor, including super-scalar implementations of the architecture.

2.15.5 Coprocessor Instructions

2.15.5.1 What Coprocessors Do

Coprocessors are alternate execution units, with register files separate from the CPU. In abstraction, the MIPS architecture provides for up to four coprocessor units, numbered 0 to 3. Each level of the ISA defines a number of these coprocessors. Coprocessor 0 is always used for system control and coprocessor 1 and 3 are used for the floating point unit. Coprocessor 2 is reserved for implementation-specific use.

A coprocessor may have two different register sets:

- Coprocessor general registers
- Coprocessor control registers

Each set contains up to 32 registers. Coprocessor computational instructions may use the registers in either set.

2.15.5.2 System Control Coprocessor 0 (CP0)

The system controller for all MIPS processors is implemented as coprocessor 0 (CP0), the **System Control Coprocessor**. It provides the processor control, memory management, and exception handling functions.

2.15.5.3 Coprocessor Load and Store Instructions

Explicit load and store instructions are not defined for CP0; for CP0 only, the move to and from coprocessor instructions must be used to write and read the CP0 registers. The loads and stores for the remaining coprocessors are summarized in “Coprocessor Loads and Stores” on page 60.

© 2008 Microchip Technology Inc.

2.16 CPU INITIALIZATION

Software is required to initialize the following parts of the device after a Reset event.

2.16.1 General Purpose Registers

The CPU register file powers up in an unknown state with the exception of r0 which is always '0'. Initializing the rest of the register file is not required for proper operation in hardware. Depending on the software environment however, several registers may need to be initialized. Some of these are:

- sp – stack pointer
- gp – global pointer
- fp – frame pointer

2.16.2 Coprocessor 0 State

Miscellaneous CP0 states need to be initialized prior to leaving the boot code. There are various exceptions which are blocked by ERL = 1 or EXL = 1 and which are not cleared by Reset. These can be cleared to avoid taking spurious exceptions when leaving the boot code.

Table 2-13: CP0 Initialization

CP0 Register	Action
CAUSE	WP (Watch Pending), SW0/1 (Software Interrupts) should be cleared.
CONFIG	Typically, the K0, KU and K23 fields should be set to the desired Cache Coherency Algorithm (CCA) value prior to accessing the corresponding memory regions.
COUNT(1)	Should be set to a known value if Timer Interrupts are used.
COMPARE(1)	Should be set to a known value if Timer Interrupts are used. The write to compare will also clear any pending Timer Interrupts (Thus, Count should be set before Compare to avoid any unexpected interrupts).
STATUS	Desired state of the device should be set.
Other CP0 state	Other registers should be written before they are read. Some registers are not explicitly writ-able, and are only updated as a by-product of instruction execution or a taken exception. Uninitialized bits should be masked off after reading these registers.

Note 1: When the Count register is equal to the Compare register a timer interrupt is signaled. There is a mask bit in the interrupt controller to disable passing this interrupt to the CPU if desired.

2.16.3 Bus Matrix

The BMX should be initialized before switching to User mode or before executing from DRM. The values written to the bus matrix are based on the memory layout of the application to be run.

 © 2008 Microchip Technology Inc.

2.17 EFFECTS OF A RESET

2.17.1 MCLR Reset

The PIC32MX core is not fully initialized by hardware Reset. Only a minimal subset of the processor state is cleared. This is enough to bring the core up while running in unmapped and uncached code space. All other processor state can then be initialized by software. Power-up Reset brings the device into a known state. Soft Reset can be forced by asserting the MCLR pin. This distinction is made for compatibility with other MIPS processors. In practice, both Resets are handled identically with the exception of the setting of StatusSR.

2.17.1.1 Coprocessor 0 State

Much of the hardware initialization occurs in Coprocessor 0.

Table 2-14: Bits Cleared or Set by Reset

Bit Name	Cleared or Set	Value	By	Cleared or Set	Value	By
StatusBEV	Cleared	1	Reset or Soft Reset			
StatusTS	Cleared	0	Reset or Soft Reset			
StatusSR	Cleared	0	Reset	Set	1	Soft Reset
StatusNMI	Cleared	0	Reset or Soft Reset			
StatusERL	Set	1	Reset or Soft Reset			
StatusRP	Cleared	0	Reset or Soft Reset			
Configuration fields related to static inputs	Set	input value	Reset or Soft Reset			
ConfigK0	Set	010 (uncached)	Reset or Soft Reset			
ConfigKU	Set	010 (uncached)	Reset or Soft Reset			
ConfigK23	Set	010 (uncached)	Reset or Soft Reset			
DebugDM	Cleared	0	Reset or Soft Reset(1)			
DebugLSNM	Cleared	0	Reset or Soft Reset			
DebugIBusEP	Cleared	0	Reset or Soft Reset			
DebugIEXI	Cleared	0	Reset or Soft Reset			
DebugSSt	Cleared	0	Reset or Soft Reset			

Note 1: Unless EJTAGBOOT option is used to boot into DEBUG mode.

2.17.1.2 Bus State Machines

All pending bus transactions are aborted and the state machines in the SRAM interface unit are reset when a Reset or Soft Reset exception is taken.

2.17.2 Fetch Address

Upon Reset/SoftReset, unless the EJTAGBOOT option is used, the fetch is directed to VA 0xBFC00000 (PA 0x1FC00000). This address is in KSeg1, which is unmapped and uncached.

2.17.3 WDT Reset

The status of the CPU registers after a WDT event depends on the operational mode of the CPU prior to the WDT event.

If the device was not in Sleep a WDT event will force registers to a Reset value.

  © 2008 Microchip Technology Inc.

2.18 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the CPU of the PIC32MX family include the following:

Title	Application Note #
No related application notes at this time.	N/A

Note:	Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

2.19 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (April 2008)

Revised status to Preliminary; Revised Section 2.1 (Key Features); Revised Figure 2-1; Revised U-0 to r-x.

Revision C (May 2008)

Revise Figure 2-1; Added Section 2.2.3, Core Timer; Change Reserved bits from “Maintain as” to “Write”.

 © 2008 Microchip Technology Inc.