PPC to MIPS® Architecture Migration Guide

Document Number: MD00875
Revision 00.10
May 30, 2011

Contents

Introduction

The purpose of this document is to describe the ease of migration from an Power ISA v.2.03 to the MIPS® architecture and cores.

MIPS Technologies supports the MIPS32® and MIPS64® instruction set architectures. MIPS64 allows 64-bit addressing modes to facilitate larger virtual address space. Migration from the MIPS32 to the MIPS64 architecture is a seamless path.

Power architecture also supports 32 and 64 bit ISA. Beyond 32-bit and 64-bit, MIPS supports 16 bit instructions for improved code density. Both architectures support floating point and several ISA extensions.

Typically the application code running on these architectures is coded in a high level language such as C or C++, so porting between architectures is straightforward. MIPS provides a GNU tool chain that that can efficiently recompile the code to a MIPS platform. The extensive ecosystem for MIPS provides a variety of operating systems, software development tools and platforms from a broad range of vendors.

The bulk of the migration effort between architectures involves low-level boot code and device initialization. The areas that need special attention are: programming model, virtual to physical address mapping differences, cache and TLB initialization, differences in exception vectors and exception types and interrupt exceptions. For assembly code translation, the user needs to understand the differences in instruction set and register calling conventions.

This document is not meant to be an architecture reference manual nor a software users guide. The purpose of this document is to illustrate the differences in areas that need special attention by the user and also provides sample code segments for initialization of the resources.

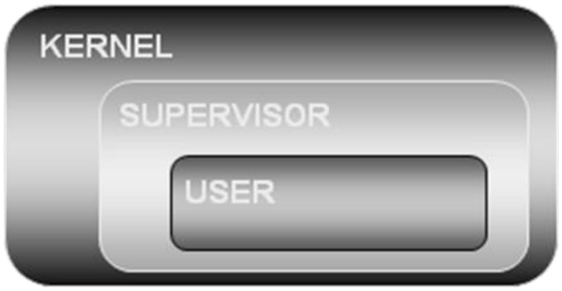
The user is encouraged to refer to the following documents for further reading and references:

MIPS32® Architecture for Programmers Volume I: Introduction to the MIPS32® Architecture
MIPS32® Architecture for Programmers Volume II: The MIPS32® Instruction Set
MIPS® Architecture for Programmers Volume III: The MIPS32® and microMIPS32™ Privileged Resource Architecture
YAMON™ Porting Requirements Specification
YAMON™ User's Manual

Details on tools and software, development kits, reference and users manuals and application notes can be found at mips.com.

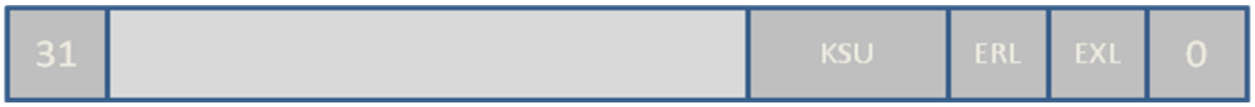
Programming Model

MIPS Security Levels:



At the kernel level all processor resources are accessible. At the supervisor level all registers, supervisor segment and the 2GB user segment are accessible. At the user level the 2GB virtual address space is accessible. Typical usages are kernel and user modes. Supervisor mode is rarely used.

The kernel, supervisor and user state selection is made via the status register.



Mode	KSU	ERL	EXL
Kernel	x x	x	1
	x x	1	x
	0 0	x	x
Supervisor	0 1	0	0
User	1 0	0	0

The Power architecture supports two modes. User mode is unprivileged and Supervisor mode is privileged. Applications normally run in user mode which is unprivileged or system mode (privileged).

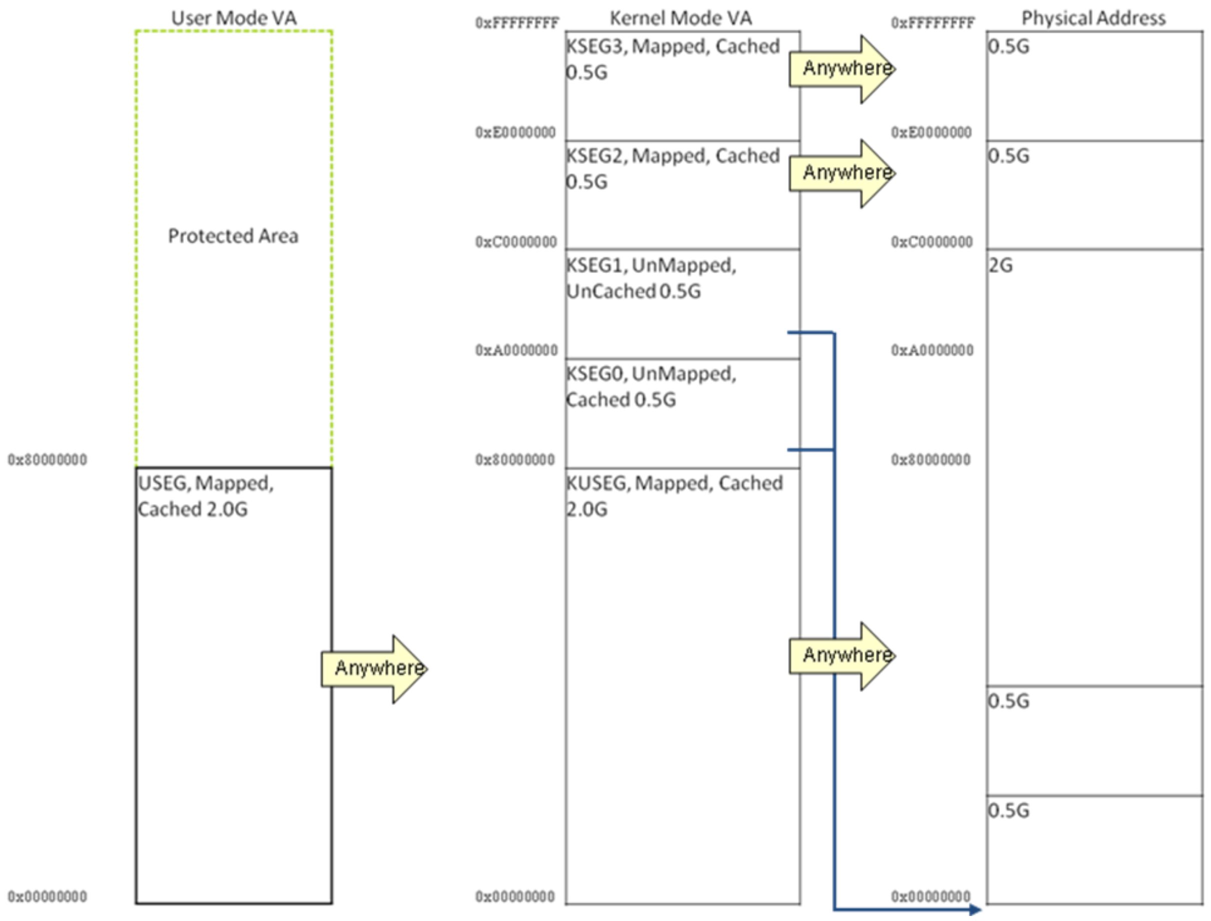
Both MIPS and PPC handle exceptions in the privileged mode.

The MIPS32 architecture specifies fixed memory map. The address space is divided into 4 regions:

- kseg2, TLB-mapped cacheable kernel space
- kseg1, direct-mapped uncached kernel space
- kseg0, direct-mapped cached kernel space
- kuseg, TLB-mapped cacheable user space

kseg0 and kseg1 segments are direct mapped and map to the first 512 megabytes of the physical address space. The rest of the regions are TLB-mapped and cacheable. Reset vector is 0xBFC00000 – kseg1. All exceptions default to kseg1 and can be relocated to kseg0 upon enabling of caches.

The following figure shows the virtual address to physical address mapping of the modes supported in MIPS cores:



Instruction set

Instruction sets for MIPS and PPC are similar. MIPS also supports application specific extensions (ASE) for DSP, security, multi-threading and other technologies. The CorExtend™ feature enables user defined instructions to be part of the core instructions set. The MIPS ISA is fully backward-compatible. The following table lists the classes of instructions both architectures support:

	PPC405Fx	**MIPS32® 24Kc™**
Add	ADD RT, RA, RB	ADD rd, rs, rt
Subtract	subf RT, RA, RB	SUB rd, rs, rt
Multiply	mulchw RT, RA, RB Rc=0	MUL rd, rs, rt
Multiply-accumulate		MADD rs, rt
Count leading zeros	cntlzw RA, RS Rc=0	CLZ rd, rs
AND	crand BT, BA, BB	ADD rd, rs, rt
XOR	creqv BT, BA, BB	XOR rd, rs, rt
OR	cror BT, BA, BB	OR rd, rs, rt
Branch	B{cond} <label>	J target
Branch with link	BL{cond} <label>	JAL target
Branch and exchange	BX{cond} <Rm>	JALX target
Word	lwz[u][x]	LW rt, offset(base)
Byte	lbz[u][x]	LBU rt, offset(base)
Byte signed		LB rt, offset(base)
Halfword	lha[u][x]	LHU rt, offset(base)
Halfword signed		LH rt, offset(base)
Word	stw[u][x]	SW rt, offset(base)
Byte	stb[u][x]	SB rt, offset(base)
Halfword	sth[u][x]	SH rt, offset(base)
Move to reg from coproc	NA	MFC0 rt, rd, sel
Move to coproc from reg	NA	MTC0 rt, rd, sel
Signed add high 16 + 16, low 16 + 16, set GE flags	NA	ADDQ.PH rd,rs,rt
Saturated add high 16 + 16, low 16 + 16	NA	ADDQ_S.PH rd,rs,rt
Signed high 16 - low 16, low 16 + high 16, set GE flags	NA	SUBQ.PH rd,rs,rt
Saturated high 16 - low 16, low 16 + high 16	NA	SUBQ_S.PH rd,rs,rt
Four saturated unsigned 8 + 8	NA	ADDU_S.QB rd, rs, rt
Four saturated 8 - 8	NA	ADD_S.QB rd, rs, rt
Four saturated unsigned 8 - 8	NA	SUBU_S.QB rd, rs, rt

CPU Initialization

The cache architecture for both the MIPS and PPC architectures are fairly similar: independent L1 for instruction and data and common a L2. MIPS caches are 1, 2 or 4 ways set associate and the line size is 4 or 8 words.

The following code segments show cache operations for a MIPS32 24Kc core and the PPC 405:

Function	PowerPC 405 CPU Core	MIPS32® 24Kc™
Enabling Cache	addis r4,r0,0x8000	mfc0 t0, C0_Config1
	addi r4,r4,0x0001	/* set kseg0 as Cacheable, noncoherent, write-back, write allocate */
	mticcr r4 /* instruction cache */	ori t0,t0, 0x3
	isync	mtc0 t0, C0_Config1
	addis r4,r0,0x0000	
	addi r4,r4,0x0000	
	mtdccr r4 /* data cache */	

Function	PowerPC 405 CPU Core	MIPS32® 24Kc™
Invalidate Cache		mfc0 a1, C0_Config1
		and a1, M_Config1IL
	/*---------------------------------- */	srl a1, S_Config1IL
	/* Invalidate I and D caches. Enable I cache for defined memory regions */	li v0, 0x2
	/* to speed things up. Leave the D cache disabled for now. It will be */	sll v0, a1 /* a1 = I-cache line size */
	/* enabled/left disabled later based on user selected menu options. */	mfc0 t9, C0_Config1
	/* Be aware that the I cache may be disabled later based on the menu */	and t8, t9, M_Config1IA
	/* options as well. See miscLib/main.c. */	srl t8, S_Config1IA
	/*------------------------------------- */	addiu t8, 1 /* t8 = associativity */
	bl invalidate_icache	and t9, M_Config1IS
	bl invalidate_dcache	srl t9, S_Config1IS
		li t7, 0x40
		sll t7, t9 /* t7 = sets per way */

		multu t8, t7 mflo a0 /* a0=cache size */ MTC0(zero, C0_TagHi) MTC0(zero, C0_TagLo) 0:00 li a2, KSEG0BASE /* Calc 1st cache line address*/ addu a3, a2, a0 /* Calc last cache line address*/ subu a3, a1 1: /* Loop through all lines, invalidating each of them */ cache ICACHE_INDEX_STORE_TAG, 0(a2) /* clear tag */ bne a2, a3, 1b addu a2, a1

TLB initialization:

Both the PPC and MIPS architectures support virtual to physical address translation via TLB scheme. On a TLB miss in the MIPS architecture, an exception is raised and an exception handler loads the appropriate configuration in the TLB. The following code segment shows TLB initialization:

Function	PowerPC 405 CPU Core	MIPS32® 24Kc™

Initilize TLB Entry	tlbwelo r4,r0
	tlbwehi r3,r0
		void initTLBEntryByIndex (int idx) {
		  int i;
		  __asm__ __volatile ("move $t0, %0" : : "r" (idx));
		  __asm__ __volatile (
		    "mtc0 $t0, $0, 0;" // set index
		    "lui $t1, 0xa000;"
		    "sll $t0, $t0, 16;"
		    "or $t1, $t0,$t1;"
		    "mtc0 $t1, $10,0;" //entryhi
		    "mtc0 $zero, $2,0;" //entrylo0
		    "mtc0 $zero, $3,0;" //entrylo1
		    "mtc0 $zero, $5,0;" //pagemask
		    "tlbwi;"
		    "ehb;"
		  );
		  return;
		}

Exception vector and exception type

The following table is a summary of the exception vector and types for the MIPS and PPC architectures:

PowerPC 405Fx CPU Core	**MIPS32® 24Kc™**
Supports different 18 types/priorities of exceptions	Supports35 different types/priorities of exceptions: offering the programmer more knowledge of what went wrong and allowing the user to handle it differently
The gap between 0xFC000000 and 0xFC900000 is used for the exception vector area and system start up logging	Exception base is predefined to 0xBFC0.0000 and 0x8000.0000 Exception base is can be changed by Ebase

The following tables list the details of exception vector, exception types and priorities for the PPC and MIPS architectures:

PPC Exception Vector Prefix Register (EVPR)is a 32-bit register whose high-order 16 bits contain the prefix for the address of an interrupt handling routine. The 16-bit interrupt vector offsets are following table	**Offset**	**MIPS Exception vector address (SI_UseExceptionBase, Status.BEV, Status.EXL, Cause.IV, EJTAG ProbEn)**	**Exception Types**
System Reset	0x00100	16#BFC0.0200	Other, TLB Refill
Machine Check	0x00200	16#BFC0.0300	Cache Error
Data Storage	0x00300	16#BFC0.0380	TLB Refill, Interrupt, All Others
Instruction Storage	0x00400	16#BFC0.0400	Interrupt
External	0x00500	16#BFC0.0480	EJTAG Debug
Alignment	0x00600	16#8000.0180	TLB Refill, Interrupt, All Others
Program	0x00700	16#8000.0280	Interrupt

Floating-Point Unavailable	0x00800	16#A000.0100	Cache Error
System Call	0x00C00	16#FF20.0200	EJTAG Debug
APU unavailable	0x00F20	EBase31..30=2#10 || 1 || EBase28..12 || EBase12..0=16#000	Cache Error
Programmable Interval Timer	0x01000	EBase31..30=2#10 || 1 || EBase28..12 || EBase12..0=16#480	EJTAG Debug
Fixed Interval Timer	0x01010	EBase31..30=2#10 || EBase29..12 || EBase12..0=16#000	All Others, Reset, NMI
Watchdog timer	0x01020	EBase31..30=2#10 || EBase29..12 || EBase12..0=16#200	All Others, TLB Refill
Data TLB miss	0x01100	EBase31..30=2#10 || EBase29..12 || EBase12..0=16#380	TLB Refill, Interrupt, All Others
Instruction TLB Miss	0x01200	EBase31..30=2#10 || EBase29..12 || EBase12..0=16#400	Interrupt
		EBase31..30=2#10 || EBase29..12 || EBase12..0=16#480	EJTAG Debug
		2#101 || SI_ExceptionBase[28:12] || 16#300	Cache Error

PowerPC 405		**MIPS32® 24Kc™**		
Priority	**Name**	**Priority**	**Name**	**Description**
1	Machine check—data	1	Reset	Assertion of SI_Reset signal.
2	Debug—IAC	2	DSS	EJTAG Debug Single Step
3	Machine check—instruction	3	DINT	EJTAG Debug Interrupt: caused by the assertion of the external EJ_DINT input, or by setting the EjtagBrk bit in the ECR register
4	Debug—EXC, UDE	4	DDBLImpr/DDBSImpr	Debug Data Break Load/Store Imprecise
5	Critical interrupt input	5	NMI	Asserting edge of SI_NMI signal
6	Watchdog timer—first time-out	6	Machine Check	TLB write that conflicts with an existing entry
7	Instruction TLB Miss	7	Interrupt	Assertion of unmasked hardware or software interrupt signal

8				
	Instruction storage — ZPR[Zn] = 00	8	Deferred Watch	Deferred Watch (unmasked by K|DM->!(K|DM) transition)
9	Instruction storage — TLB_ entry, Program ,System call, APU Unavailable, FPU Unavailable	9	DIB	EJTAG debug hardware instruction break matched
10		10	WATCH	A reference to an address in one of the watch registers (fetch)
11	Data TLB miss	11	AdEL	Fetch address alignment error: fetch reference to protected address
12	Data storage	12	TLBL	Fetch TLB miss: fetch TLB hit to page with V=0
13	Data storage	13	ICache Error	Parity error on ICache access
14	Alignment	14	IBE	Instruction fetch bus error
15	Debug	15	DBp	EJTAG Breakpoint (execution of SDBBP instruction)
16	External interrupt input	16	Sys	Execution of SYSCALL instruction.
17	Fixed Interval Timer	17	Bp	Execution of BREAK instruction
18	Programmable Interval Timer	18	CpU	Execution of a coprocessor instruction for a coprocessor that is not enabled
		19	CEU	Execution of a CorExtend instruction modifying local state when CorExtend is not enabled
		20	RI	Execution of a Reserved Instruction
		21	FPE	Floating Point exception

		22	C2E	Coprocessor2 Exception
		23	IS1	Implementation specific Coprocessor2 exception
		24	Ov	Execution of an arithmetic instruction that overflowed
		25	Tr	Execution of a trap (when trap condition is true)
		26	DDBL / DDBS	EJTAG Data Address Break (address only)
		27	WATCH	A reference to an address in one of the watch registers (data)
		28	AdEL	Load address alignment error. Load reference to protected address
		29	AdES	Store address alignment error. Store to protected address
		30	TLBL	Load TLB miss. Load TLB hit to page with V=0
		31	TLBS	Store TLB miss. Store TLB hit to page with V=0
		32	TLB Mod	Store to TLB page with D=0.
		33	DCache Error	Cache parity error - imprecise
		34	L2 Cache Error	L2 Cache ECC error - imprecise
		35	DBE	Load or store bus error - imprecise

Interrupt exception

MIPS architecture has an integrated interrupt controller that supports up to 6 priorities in VI mode and up to 63 priorities in EIC mode. The PPC supports two external interrupt sources, critical and non-critical. The following table summarizes the interrupt schemes for each architecture:

PowerPC 405	MIPS32® 24Kc™
Two types Critical and Noncritical Interrupts	One type but has 3 operation mode (Compatibility, VI, EIC)
Bank registers for supporting better interrupt response	Shadow register for better interrupt service response
Max 32 active registers can be used	Max 32 active registers can be used
Need to handshake with external interrupt controller to get effective vector address	Faster interrupt response time as CPU calculates the vector address
Need to reload the PC for both critical and non-critical interrupts	The spacing is set 0x20 and the base 0x8000.280. Shadow registers can be bound to different interrupt sources

PowerPC 405		MIPS32® 24Kc™	
Exception Address	Critical / Noncritical Handler	Exception Address	Interrupt Handler
#Handler Addr1	Critical handler ... Include code to process the interrupt Store address of next to SRR2 Store contents of MSR (status) to SRR3 ... rfci	0x8000.02A0	Interrupt Handler 1 ... Include code to process the interrupt ... ... ERET
#Handler Addr2	Noncritical handler (For example: external interrupt) ... Include code to process the interrupt Store address of next to SRR0 Store content of MSR to SRR1 ... rfi	0x8000.02C0	Interrupt Handler 2 ... Include code to process the interrupt ... ... ERET

Application Binary Interface (ABI)

The following table shows the register calling convention for the PPC and MIPS architectures:

PowerPC 405 core			MIPS 24Kc		
`r0`	local	commonly used to hold the old link register when building the stack frame	$0	$0	Always 0

`r1`	dedicated	stack pointer	$1	$at	The Assembler Temporary used by the assembler in expanding pseudo-ops
`r2`	dedicated	table of contents pointer	$2-$3	$v0-$v1	These registers contain the Returned Value of a subroutine; if the value is 1 word only $v0 is significant
`r3`	local	commonly used as the return value of a function, and also the first argument in	$4-$7	$a0-$a3	The Argument registers, these registers contain the first 4 argument values for a subroutine call
`r4-r10`	local	commonly used to send in arguments 2 through 8 into a function	$8-$15, $24,$25	$t0-$t9	The Temporary Registers
`r11-r12`	local		$16-$23	$s0-$s7	The Saved Registers
`r13-r31`	global		$26-$27	$k0-$k1	The Kernel Reserved registers. DO NOT USE
`lr`	dedicated	link register; cannot be used as a general register. Use `mflr` (move from link register) or `mtlr` (move to link register) to get at, e.g., `mtlr r0`	$28	$gp	The Global Pointer used for addressing static global variables. For now, ignore this
`cr`	dedicated	condition register	$29	$sp	The Stack Pointer
			$30	$fp (or $s8)	The Frame Pointer: programs that do not use an explicit frame pointer (e.g., everything assigned in ECE314) can use register $30 as another saved register - not recommended however
			$31	$ra	The Return Address in a subroutine call

Migrating applications

Reset, initialization and exception handling are typically done in assembly, but it is common that the application itself is coded in C/C++ (high level language). The application and any device drivers must be recompiled with the MIPS tool chain. Any assembly code can be translated manually, as there is almost a one-to-one equivalent instruction.

The bulk of the effort in migration entails changes to the initialization and low level boot code. MIPS Technologies provides the YAMON™ PROM monitor as reference code that runs on

MIPS development boards. There are boot loaders available from third party vendors and open source.

On both the PPC and MIPS architectures, applications normally will run in user mode. On an exception the PPC will switch to the supervisor mode and the MIPS to the kernel mode. The exception handler switches the mode back to user mode upon handling the exception.

Memory map on the MIPS architecture is fixed and user space is kuseg segment of the memory. Kseg0-3 is reserved for kernel. Cached system data resides in kseg0 and uncached in kseg1. I/O devices in the MIPS architecture are mapped in kseg1. The PPC architecture does not specify a fixed memory map.

In both PPC and MIPS most of the exception handling is done in software. The MIPS architecture supports six external interrupts and each can be masked independently. Nested interrupts are handled in a similar fashion. The exception context needs to be saved for nested interrupts before re-enabling the interrupts. Configuring interrupts is straight forward in the MIPS architecture as only the status register fields must be programmed.

The MIPS ISA defines both MIPS32 and MIPS64. MIPS64 offers larger virtual address and physical address space, and MIPS32 applications can be seamlessly migrated to MIPS64 to take advantage of the 64-bit pointers. Long word is 128 bits in MIPS64 and in both MIPS32 and MIPS64, char is 8-bit unsigned. MIPS provides N32 and N64 ABIs for embedding assembly code with C/C++.

The MIPS architecture load and store instructions require that all data is aligned on its “natural” boundary, i.e shorts on a multiple of 2 bytes, ints on a multiple of 4, and doubles on 8. If the alignment is not correct, then the CPU will generate an address exception. gcc will normally align all data structures and their fields on their natural boundaries. However, some software ported from 8 or 16-bit CPUs may rely on data structures whose fields align to a smaller boundary.

There are two ways to convince gcc to change its default alignment rules:

1. Use the GCC attribute (packed)extension on whole structures or individual structure fields - see the Extensions section of the GCC manual for full details.

2. Precede the definitions of packed structures with the single line #pragma pack(x), where x is the alignment boundary, in bytes. Follow the declaration with the line #pragma pack(x), which restores the normal alignment rules - don't forget this, your code may continue to work, but quietly become bigger and slower! For example:

```
#pragma pack(1)
struct packedstruct {
short s; /* offset: 0 */
int i; /* offset: 2 */
int j; /* offset: 6 */
};
#pragma pack()
```

MIPS Technologies offers the MIPS Navigator™ Integrated Component Suite (ICS) that comes with a GNU compiler and debugger, JTAG level debugger, profiler and event analyzer. Several other software vendors also provide tool suite for MIPS processors.

Summary

MIPS Technologies licenses its MIPS32 and MIPS64 architectures, and also offers single-core, multi-core, superscalar and multi-threaded families of cores based on the MIPS32 architecture. Several of MIPS Technologies' licensees also offer high-performance, multicore products based on the MIPS64 architecture.

With its architectures and cores, MIPS has a large footprint in digital home and networking applications, and growing traction in mobile devices.

MIPS cores are the industry's most area efficient, offering high performance at the lowest power dissipation. With its multi-threading technology, companies can efficiently implement a parallelizable application by maximizing the instructions per cycle (IPC). The QoS features in the multi-threaded family of products help ensure real-time application performance. The breadth and the rich features of MIPS' product portfolio, coupled with a flexible business model, enable MIPS licensees to create MIPS-Based products that range from 32-bit microcontrollers and energy efficient mobile devices to 'green' supercomputers and high end networking infrastructure.