microMIPS™ Instruction Set Architecture

Uncompromised Performance, Minimum System Cost

October 2009

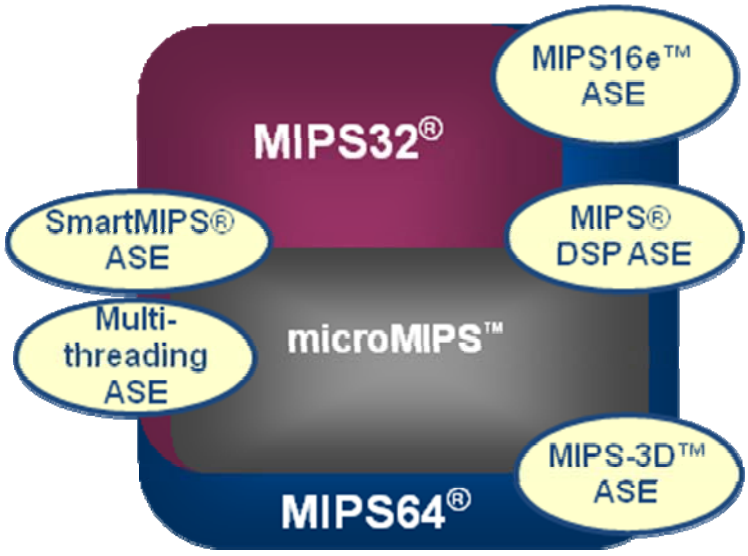
MIPS Technologies, Inc.
955 East Arques Avenue
Sunnyvale, CA 94085
(408) 530-5000

The transition to digital representation of all media – music, photos, video – has completely transformed the home. Where we once had analog TV tuners and CD-centric music systems, we now have digital set-top boxes, media players that stream Internet content, and home network gear that would have found its only use in the enterprise a few years ago. Engineering teams contemplating the design of a connected device are faced with an incredibly tough balancing act. Content such as digital video requires uncompromised performance. Meanwhile the consumer demands low cost and high quality. The only answer is a media-capable microprocessor at the core of the product that is architected for low system cost, low power, and application acceleration. The new microMIPS™ architecture delivers uncompromised, 32-bit performance along with the memory footprint of a 16-bit device – minimizing system memory and ultimately silicon and system cost.

The microMIPS architecture is the latest in an evolving family of architectures from MIPS Technologies. The term 'architecture' essentially refers to the set of instructions, or the ISA (Instruction Set Architecture) that a processor can execute. Of course, someone must design the processor to handle the ISA. And MIPS Technologies is delivering two microMIPS-compatible processor cores – the MIPS32® M14K™ and M14Kc™ cores – in the form of Intellectual Property (IP).

The MIPS® architecture originated in the early 1980s and was targeted at the workstation market. MIPS was one of the definitive Reduced Instruction Set Computer (RISC) architectures. RISC processors generally rely on a load/store architecture to move data from memory to registers. Moreover, the architectures generally include hardwired logic rather than microcode to execute instructions, thereby handling instructions in a single clock cycle. The MIPS architecture has stayed true to the RISC tenets over the years while adding to and enhancing the original ISA.

Today, MIPS Technologies' primary ISAs are MIPS32 and MIPS64® – 32- and 64-bit architectures respectively. The company has augmented the basic ISAs with extensions called Application Specific Extensions (ASEs) that target specific applications. For instance, the MIPS DSP ASE boosts multimedia performance for consumer-centric chip designs.



MIPS also previously developed an extension called MIPS16e™ that was designed to minimize code size and thus the memory required in an application. MIPS16e introduced a number of 16-bit instructions that could be used in place of the standard 32-bit instructions with the goal of minimizing system cost. We'll get to the specifics of the new microMIPS ISA shortly, and how it improves on the MIPS16e effort.

Before we get to microMIPS details, let's consider the target application. A product such as a video-enabled set-top box must, at a minimum, be capable of decoding digital video in real time. Some products such as the venerable TiVo also encode video in real time. Both are performance-intensive tasks, although the encode task is decidedly more difficult.

Based on the application at hand, system designers can use dedicated encoder/decoder (codec) ICs to handle video. A dedicated IC might be the only option for high-definition formats. But for lower-fidelity devices with smaller screens, a general-purpose processor can execute a software codec.

Programmable processors bring great benefits to multimedia-centric system designs. Codec standards are constantly evolving for both audio and video. So long as a processor can handle the codec task, the software option is generally less expensive than a dedicated IC. And with a programmable processor, the software codec can evolve with a changing standard. Even if a dedicated video codec is required in a system, design teams can handle the audio codec in software and add features such as support for multiple codecs. Because MIPS delivers architectures in core form, design teams can also develop SoCs that combine the processor and dedicated functions such as codecs.

Despite the performance requirements, multimedia consumer products and networking gear must be low cost. Traditionally, people think of memory as cheap. And in the context of even a $500 PC, a gigabyte of memory is relatively cheap. Consumer electronics, however, sell for less. And even savings of $10 or $20 in a bill of materials

(BOM) can make or break a product. Reducing memory requirements is a key way to cut BOM costs.

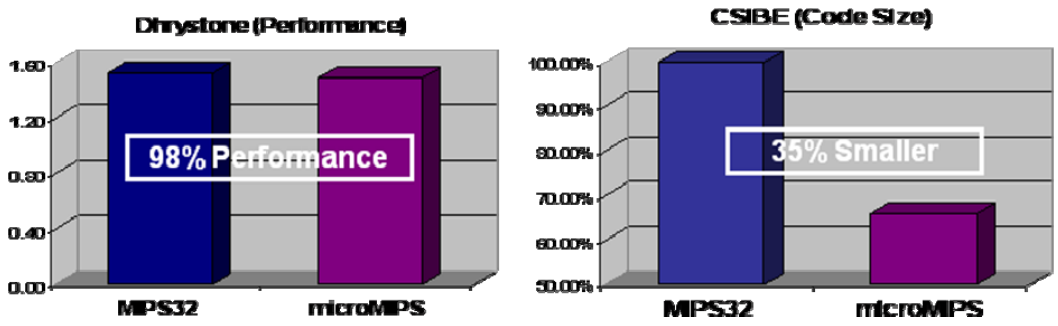
The MIPS32 architecture offers the performance needed for media-centric applications, but does so with the code size typical of 32-bit processors. Given the target markets of low-footprint consumer electronics and connected appliances, the microMIPS design team set out to develop a new architecture that could deliver on several key requirements:

Uncompromised performance in the MIPS32 class
A code size reduction of 30% or more and equivalent memory reduction
Support for existing MIPS32 and MIPS64 ISAs
Maintain the high-performance microarchitecture and legacy MIPS32 support

The microMIPS architecture and core designs deliver on all of these goals. The new architecture is assembly-language compatible with existing source code, and offers compatibility at the ABI (Application Binary Interface) level that defines the interface between applications and operating systems. A microMIPS-based design won't achieve code size reduction on existing MIPS32 code, but programmers can easily recode assembly language or recompile high-level code to take advantage of microMIPS.

Benchmarks

MIPS engineers have run a series of benchmarks testing both performance and code size. In the CSiBE code-size benchmark, microMIPS delivers a 35% reduction relative to MIPS32. And in the Dhrystone performance benchmark, microMIPS delivers 98% of the performance of MIPS32.



MIPS engineers ran the benchmarks on both simulations of the microMIPS ISA and on the RTL (Register Transfer Level) representation of the M14K and M14Kc cores that will be handed off to licensees working on IC designs.

The benchmarks prove that the microMIPS architecture delivers similar memory savings as the MIPS16e ASE, but with much better performance. The question is how? The list of

reasons includes new optimized 16- and 32-bit instructions, an optimized recoding of MIPS32 instructions, and optimized op code format and register utilization.

The earlier stated fact that microMIPS is a complete architecture – not an extension − is also key. Both MIPS16e, and for that matter the similarly-targeted ARM Thumb technologies, are extensions. Both microMIPS and these extensions rely on the concept of new 16-bit versions of regularly used instructions to minimize code size. And all deliver reductions in code size. But only microMIPS delivers 32-bit performance.

microMIPS Performance

To understand the significance of the new architecture, consider how programmers use the 16-bit instructions. With MIPS16e or Thumb, the programmer must expressly swap into 16-bit mode to use the new instructions. But the new instructions are a subset of the overall 32-bit ISAs. Moreover, the MIPS16e and Thumb designs must use the normal 32-bit mode to execute privileged instructions and therefore exception handling. The 32-bit mode is also requisite for floating-point instructions and other regularly occurring operations. The mode switch to and from 16-bit mode adds performance overhead.

The microMIPS ISA offers the most commonly used instructions in a 16-bit format as well as all of the instructions from the MIPS32/64 ISAs. In effect, microMIPS offers two ISAs in one architecture. As with MIPS16e, the programmer can swap between MIPS32 and microMIPS modes. But for code written expressly for the new microMIPS ISA, there's never a need for a mode swap.

The microMIPS mode can handle all operations such as exception handling, and offers a superset of the MIPS32 ISA. With MIPS16e, the programmer had to swap modes to use ASEs such as MIPS DSP. The microMIPS mode can seamlessly access the ASEs. The need to run legacy binary code is really the only reason that would prompt a programmer to use the MIPS32 mode.

In addition to eliminating mode switches, the microMIPS ISA includes several other performance enhancements. The ISA includes new instructions that boost performance and reduce code size. For example, the ISA includes both 16- and 32-bit instructions that perform multiple load or store operations with a single instruction. The 16-bit Load Word Multiple (LWM16) instruction can move data from consecutive memory locations to as many as five registers, while the 32-bit version can load nine registers with a single instruction.

The microMIPS ISA also eliminates an important shortcoming of MIPS16e − the MIPS16e ASE could only access 8 of 32 GPRs (General Purpose Registers). Some of the 16-bit microMIPS instructions – such as arithmetic instructions – are also limited to eight GPRs. But Move instructions in microMIPS can access all 32 GPRs, and obviously all 32-bit instructions can reach all GPRs. Finally, microMIPS provides hardware support for instructions that are misaligned due to the mix of 16- and 32-bit instructions.

In total, the microMIPS ISA adds 54 new instructions – a mix of 16- and 32-bit instructions. The rest of the ISA is comprised of instructions from the MIPS32 and MIPS64 ISAs – although the microMIPS implementation remaps op codes. You will find a more detailed description of the architecture a bit later in this whitepaper.

Cores and Development Tools

MIPS customers can immediately start microMIPS based designs using MIPS' new M14K and M14Kc cores. The base-level M14K core targets typical microcontroller applications. The M14Kc adds some features and targets home entertainment and home networking products. The enhancements in the M14Kc include cache that boosts performance and a Memory Management Unit (MMU) that allows demand-paging operating systems such as Linux to run on the processor.

System design teams can start development leveraging MIPS' new SEAD-3 SoC Evaluation and Development Platform – essentially the same board used in the performance benchmarks. The 8x7.5-inch board includes 64 Mbytes of flash memory and 4 Mbytes of 32-bit-wide SRAM. Two versions offer a choice of LX50 or LX110 Xilinx FPGAs. The latter offers more I/O in the FPGA and a DDR II SDRAM socket. The boards also include expansion connector, MicroSD card slot and GPIO, I^2C, SPI, and ADC interfaces.

For software development, MIPS offers a comprehensive suite of tools for microMIPS. The offerings include the CodeSourcery Sourcery G++ (SG++) suite which includes the Eclipse Integrated Development Environment (IDE) and complete GNU toolchain including C and C++ compilers, linker, assembler, and debugger. SG++ is available for both Bare Iron/RTOS and Linux targets. MIPS also offers the MIPS Navigator™ Integrated Component Suite (ICS), an Eclipse-based IDE that integrates the SG++ toolchain, fully supports the MIPS System Navigator™ probe and provides support for debug plug-ins such as the Hot Spot Analyzer and Linux Event Analyzer.

Microarchitecture and ISA Details

Now let's dig deeper into the microarchitecture that underlies microMIPS, examples of instructions, and a look at the instruction word and op code formats.

The microMIPS-based M14K and M14Kc cores implement the same high-performance 5-stage pipeline used in the MIPS32 M4K® family microarchitecture – delivering 1.5 DMIPS/MHz Dhrystone performance. The real difference comes in the instruction-decode stage, or I-Stage at the beginning of the pipeline. The instruction decoder in a microMIPS implementation performs two operations in sequence. First, the decoder translates or recodes the microMIPS instruction into a MIPS32 instruction. Then the decoder handles the MIPS32 instruction just as in any other design.

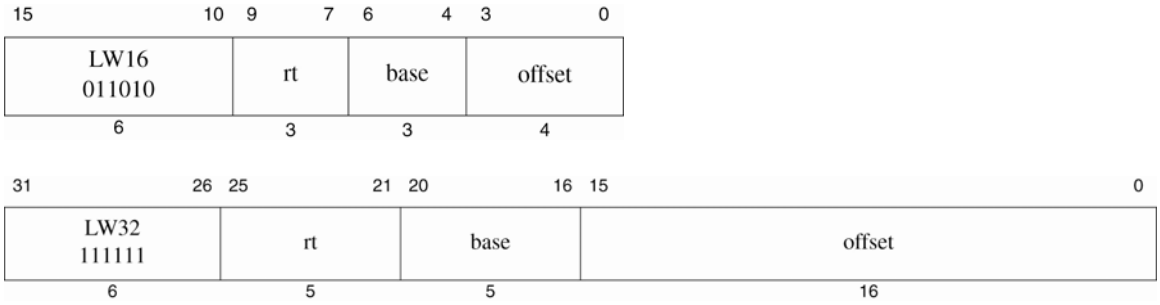
The recode and decode implementation offers many advantages. The microMIPS team was assured of realizing the performance of a proven MIPS32 microarchitecture. When

in MIPS32 mode, the instruction flow simply bypasses the recode step. Because the operation is implemented in the instruction decoder, the memory savings realized by 16-bit instructions apply both to main memory and cache.

The serial instruction-decode architecture delivers both microMIPS as a new optimized ISA and the ability to support the MIPS32/64 ISAs. And the architecture clearly differentiates microMIPS and MIPS16e. The latter used a 16-bit instruction decoder that operated in parallel with the primary instruction decoder. As mentioned earlier, programmers had to explicitly swap into 16-bit mode and there was no access to the privileged instructions needed for exception handling in 16-bit mode. The microMIPS ISA is optimized and complete with full access to privileged instructions and no required mode switches.

Paired 16- and 32-bit microMIPS Instructions

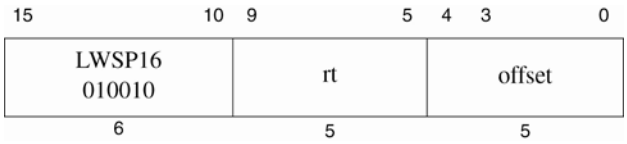
Now let's have a detailed look at some microMIPS and MIPS32 instructions to understand the new ISA and how it impacts code size and performance. The MIPS32 ISA includes a Load Word (LW) instruction. The microMIPS ISA includes 16- and 32-bit Load Word) instructions (LW16 and LW32). The 32-bit LW and LW32 instructions execute in an identical fashion, although the microMIPS LW32 instruction did get a new op code. The microMIPS LW16 is a new instruction. The instruction word diagrams below provide details.



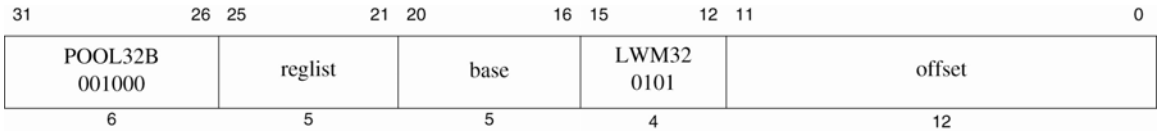
The instructions load data from memory that's pointed to by a base address stored in a GPR added to an offset value, and stores the value into the target GPR (rt). As you can see, the base and target fields in the 16-bit versions can only access 8 of the GPRs while the 32-bit version can access all 32 GPRs. And the offset in the 16-bit version is 4 bits in length rather than 16. But the 16-bit version gets an effective offset of 6 bits because, at execution time, the specified offset is left shifted by two bits with zeros inserted in the least significant bits.

Other new instructions offer a greater offset range and access to all 32 registers. For instance, the Load Word from Stack Pointer (LWSP16) instruction doesn't need a base GPR field because the use of the stack pointer as the base is implied. That leaves room in the op code for a 5-bit target register field – reaching all 32 GPRs. And the 5-bit offset

field is again left shifted by 2 bits to effectively provide a 7-bit offset value. See the instruction word below.

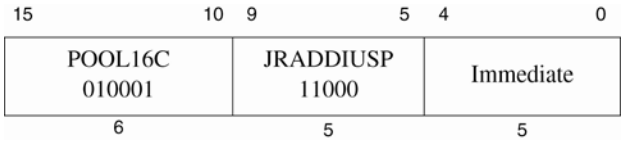


Let's also consider an instruction that can impact both code size and performance. Earlier we mentioned the Load Word Multiple instructions – new 16- and 32-bit (LWM16 and LWM32) instructions introduced with the microMIPS ISA. These new instructions have a direct impact on code size because a single instruction loads multiple registers from sequential memory locations.



Above, you see the 32-bit version of the LWM instruction. We'll explain the POOL32 op code shortly, but for now, focus on the instruction. The 5-bit reglist field specifies both which and how many registers get loaded with data during execution. One to nine GPRs are loaded from successive memory locations pointed to by the contents of the base GPR added to the offset.

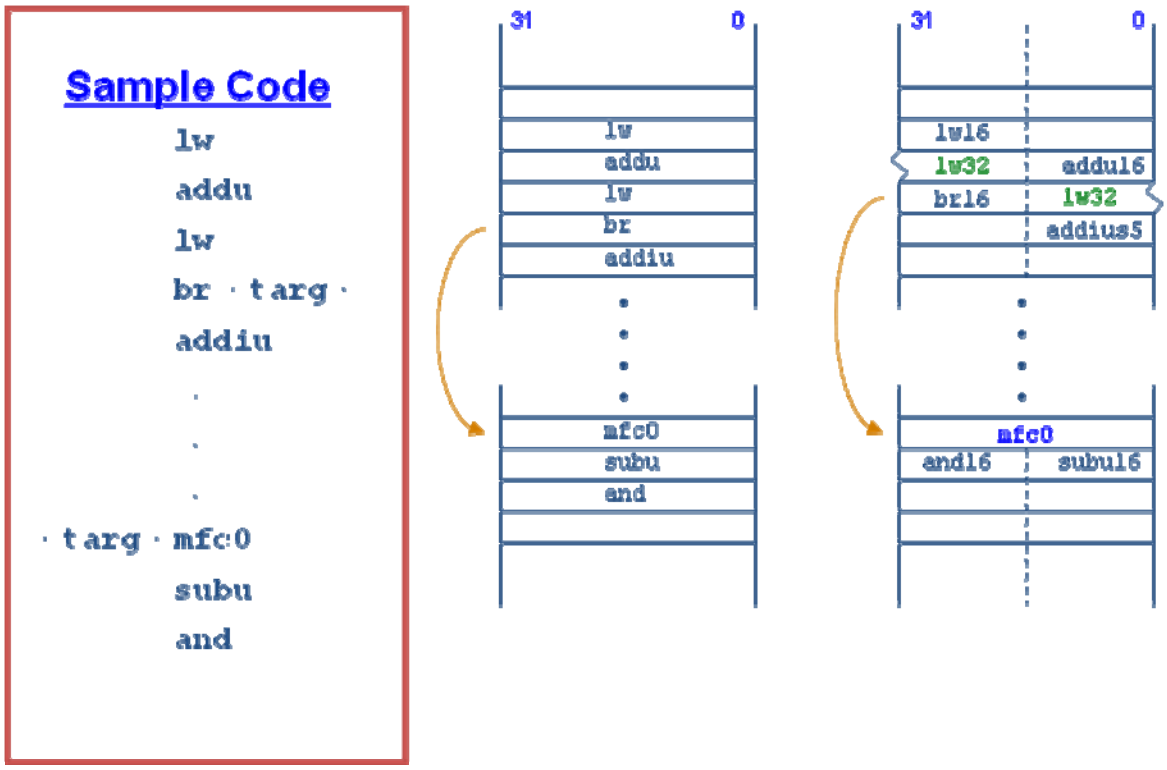
Other new instructions also directly reduce code size. Consider the 16-bit Jump Register, Adjust Stack Pointer (JRASSIUSP) instruction. This single 16-bit instruction both transfers execution to an address stored in GPR 31, and applies and modifies the Stack Pointer (GPR 29) by adding a 7-bit immediate value to the Stack Pointer. The single instruction replaces two instructions commonly used when returning from a subroutine call. The instruction word below demonstrates how much can be accomplished with 16 bits when the jump target and new SP are predetermined based on GPR contents.



All of the microMIPS instructions use a 6-bit primary or major op code, and that op code is stored in the top 6 bits of the instruction word. Some instructions are grouped or pooled together using the same 6-bit major op code and a secondary or minor op code. For example, the 32-bit Load Word Pair (LWP) and Load Word Multiple (see two recent examples above for minor op code fields) instructions use the same 6-bit major op code, but different minor op codes. In 32-bit instructions, the minor op code is located in the upper 4 bits of the lower half (lower 16 bits) of the instruction word. In the 16-bit case, the minor op code follows the major op code. The pooled op codes save space for future extensions by not using every possible 6-bit code.

microMIPS Coding Example

To understand the advantages that microMIPS offers in terms of code size, let's examine a short code sequence implemented both in MIPS32 mode and in microMIPS mode using 16-bit instructions. Consider the sequence of instructions in the diagram below. And remember that the memory sequence is right to left rather than left to right.



The LW instruction fetches a value from memory. A 16-bit instruction is used in the microMIPS example. The ADDU instruction adds two unsigned words. The second LW instruction is a 32-bit instruction in both cases. The example depicts how the instruction is split across two memory words. Still, the succeeding 16-bit branch instruction in the microMIPS example fits in the top 16-bits of the memory word that also contains the top-16 bits of the 32-bit LW instruction. That example shows a savings of one memory word. Likewise, 16-bit versions of the AND and subtract (SUBU) instructions after the branch illustrate another saved memory word.

Summary

The microMIPS ISA offers programmers, and more importantly compilers, the best of 16- and 32-bit architectures. When needed for larger offsets or larger immediate values, the compiler will automatically use the 32-bit options. When a smaller offset or access to a subset of registers will suffice, the compiler will use the 16-bit version and automatically deliver code savings. The keys to minimum code size and maximum performance are:

New optimized ISA
Proven high-performance microarchitecture
New 16-bit and 32-bit instructions
Most commonly used instructions available in 16-bit version
No mode switch required for 16-bit instructions
Hardware support for misaligned instructions and branches

The microMIPS architecture delivers on performance and code size for new applications, and backward compatibility for legacy applications.