MIPS architecture

From Wikipedia, the free encyclopedia

***MIPS** (originally an acronym for **Microprocessor without Interlocked Pipeline Stages**) is a RISC microprocessor architecture developed by MIPS Technologies. By the late 1990s it was estimated that one in three RISC chips produced were MIPS-based designs.*

MIPS designs are currently primarily used in many embedded systems such as the Series2 TiVo, Windows CE devices, Cisco routers, Foneras, Avaya, and video game consoles like the Nintendo 64 and Sony PlayStation, PlayStation 2, and PlayStation Portable handheld system. Until late 2006 they were also used in many of SGI's computer products.

The early MIPS architectures were 32-bit implementations (generally 32-bit wide registers and data paths), while later versions were 64-bit implementations. Multiple revisions of the MIPS instruction set exist, including MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, and MIPS64. The current revisions are MIPS32 (for 32-bit implementations) and MIPS64 (for 64-bit implementations). MIPS32 and MIPS64 define a control register set as well as the instruction set. Several "add-on" extensions are also available, including MIPS-3D which is a simple set of floating-point SIMD instructions dedicated to common 3D tasks, MDMX(MaDMaX) which is a more extensive integer SIMD instruction set using the 64-bit floating-point registers, MIPS16e which adds compression to the instruction stream to make programs take up less room (allegedly a response to the Thumb encoding in the ARM architecture), and the recent addition of MIPS MT, new multithreading additions to the system similar to HyperThreading in the Intel's Pentium 4 processors.

Computer architecture courses in universities and technical schools often study the MIPS architecture. The design of the MIPS CPU family greatly influenced later RISC architectures such as DEC Alpha.

Contents

History

RISC Pioneer

In 1981, a team led by John L. Hennessy at Stanford University started work on what would become the first MIPS processor. The basic concept was to dramatically increase performance through the use of deep instruction pipelines, a technique that was well known, but difficult to implement. CPUs are built up from a number of dedicated subunits known as modules or units. Typical modules include the load/store unit which handles external memory, the ALU which handles basic integer math and logic, or the FPU that handles floating point math. In a traditional design, each instruction flows from unit to unit until it is complete, at which point the next instruction is read in and the cycle continues. Generally in a pipeline architecture, successive instructions in a program sequence will overlap in execution. Instead of waiting for the instruction to complete, each unit inside the CPU will fetch and start executing an instruction before the preceding instruction is complete. For instance, as soon as a math instruction fed into the floating point module, the load/store unit can start loading up the data needed by the next instruction.

One major barrier to pipelining was that not all instructions can be handed off in this fashion. Some instructions, like a floating point division, take longer to complete and the CPU has to wait before passing the next instruction into the system. The normal solution to this problem was to use a series of interlocks that allowed the modules to indicate they were still busy, pausing the other modules upstream. Hennessy's team viewed this interlocks as a major performance barrier moving forward; since they had to communicate to all the modules in the CPU, communications time was an issue and this appeared to limit increases in clock speed. A major aspect of the MIPS design was to fit every sub-phase (including memory access) of all instructions into one cycle, thereby removing any needs for interlocking, and permitting a single cycle throughput.

Although this design eliminated a number of useful instructions, notably things like multiply and divide which would take multiple execution steps, it was felt that the overall performance of the system would be dramatically improved because the chips could run at much higher clock rates. This ramping of the speed would be difficult with interlocking involved, as the time needed to set up locks is as much a function of die size as clock rate: adding the needed hardware might actually slow down the overall speed. The elimination of these instructions became a contentious point. Many observers claimed the design (and RISC in general) would never live up to its hype. If one simply replaces the complex multiply instruction with many simpler additions, where is the speed increase? This overly-simple analysis ignored the fact that the speed of the design was in the pipelines, not the instructions.

The other difference between the MIPS design and the competing Stanford RISC involved the handling of subroutine calls. RISC used a technique called register windows to improve performance of these very common tasks, but in using hardware to do this they locked in the number of calls that could be supported. Each subroutine call required its own set of registers, which in turn required more real estate on the CPU and more complexity in its design. Hennessy felt that a careful compiler could find free registers without resorting to a hardware implementation, and that simply increasing the number of registers would not only make this simple, but increase the performance of all tasks.

In other ways the MIPS design was very much in keeping with the overall RISC design philosophy. To improve overall performance, RISC designs reduce the number of instructions in order to use fewer bits to encode them - in the MIPS design the instructions normally require only 5 bits of the 32-bit word. The rest of the space in the instruction word are used as storage, either for pointers to addresses in main memory, or as direct storage for small numbers. This allows a RISC CPU to load up the instruction and the data it needs in a single operation, whereas older designs, the MOS Technology 6502 for instance, would require separate cycles to load the instructions and data. This change is one of the major performance improvements that RISC offers.

In 1984 Hennessy was convinced of the future commercial potential of the design, and left Stanford to form MIPS Computer Systems. They released their first design, the **R2000**, in 1985, improving the design as the **R3000** in 1988.

These 32-bit CPUs formed the basis of their company through the 1980s, used primarily in SGI's series of workstations. These commercial designs deviated from the Stanford academic research by implementing most of the interlocks in hardware, supplying full multiply and divide instructions (among others).

In 1991 MIPS released the first 64-bit microprocessor, the **R4000**. However, MIPS had financial difficulties while bringing it to market. The design was so important to SGI, at the time one of MIPS' few major customers, that SGI bought the company outright in 1992 in order to guarantee the design would not be lost. As a subsidiary of SGI, the company became known as MIPS Technologies.

Licensable Architecture

In the early 1990s MIPS started licensing their designs to third-party vendors. This proved fairly successful due to the simplicity of the core, which allowed it to be used in a number of applications that would have formerly used much less capable CISC designs of similar gate count and price **--** the two are strongly related; the price of a CPU is generally related to the number of gates and the number of external pins. Sun Microsystems attempted to enjoy similar success by licensing their SPARC core but was not nearly as successful. By the late 1990s MIPS was a powerhouse in the embedded processor field, and in 1997 the 48-millionth MIPS-based CPU shipped, making it the first RISC CPU to outship the famous 68k family. MIPS was so successful that SGI spun-off MIPS Technologies in 1998. Fully half of MIPS' income today comes from licensing their designs, while much of the rest comes from contract design work on cores that will then be produced by third parties.

In 1999 MIPS formalized their licensing system around two basic designs, the 32-bit **MIPS32** (based on MIPS II with some additional features from MIPS III, MIPS IV, and MIPS V) and the 64-bit **MIPS64** (based on MIPS V). NEC, Toshiba and SiByte (later acquired by Broadcom) each obtained licenses for the MIPS64 as soon as it was announced. Philips, LSI Logic and IDT have since joined them. Success followed success, and today the MIPS cores are one of the most-used "heavyweight" cores in the marketplace for computer-like devices (hand-held computers, set-top boxes, etc.), with other designers fighting it out for other niches. Some indication of their success is the fact that Freescale (spun-off by Motorola) uses MIPS cores in their set-top box designs, instead of their own PowerPC-based cores.

Since the MIPS architecture is licensable, it has attracted several processor start-up companies over the years. One of the first start-ups to design MIPS processors was Quantum Effect Devices (see next section). The MIPS design team that designed the **R4300** started the company SandCraft, which designed the **R5432** for NEC and later produced the **SR71000**, one of the first out-of-order execution processors for the embedded market. The original DEC StrongARM team eventually split into two MIPS-based start-ups: SiByte which produced the **SB-1250**, one of the first high-performance MIPS-based systems-on-a-chip (SOC); while Alchemy Semiconductor (later acquired by AMD) produced the **Au-1000** SoC for low-power applications. Lexra used a MIPS-*like* architecture and added DSP extensions for the audio chip market and multithreading support for the networking market. Due to Lexra not licensing the architecture, two lawsuits were started between the two companies. The first was quickly resolved when Lexra promised not to advertise their processors as MIPS-compatible. The second (about MIPS patent 4814976 for handling unaligned memory access) was protracted, hurt both companies' business, and culminated in MIPS Technologies giving Lexra a free license and a large cash payment.

Two companies have emerged that specialize in building Multi-core devices using the MIPS architecture. Raza Microelectronics Inc purchased the product line from failing Sandcraft and later produced devices that contained 8 CPU cores that were targeted at the telecom and networking markets. Cavium Networks, originally a security processor vendor also produced devices with 8 CPU cores for the same markets. Both of these companies designed their cores in-house, just licensing the architecture instead of purchasing cores from MIPS.

Losing the Desktop

Among the manufacturers which have made computer workstation systems using MIPS processors are SGI, MIPS Computer Systems, Inc., Whitechapel Workstations, Olivetti, Siemens-Nixdorf, Acer, Digital Equipment Corporation, NEC, and DeskStation. Operating systems ported to the architecture include SGI's IRIX, Microsoft's Windows NT (until v4.0), Windows CE, Linux, BSD, UNIX System V, SINIX and MIPS Computer Systems' own RISC/os.

There was speculation in the early 1990s that MIPS, and other powerful RISC processors would overtake the Intel IA32 architecture. This was encouraged by the support of the first two versions of Microsoft's Windows NT for DEC Alpha, MIPS and PowerPC - and to a lesser extent the Clipper architecture and SPARC. However, as Intel quickly released faster versions of their Pentium class CPUs, Microsoft Windows NT v4.0 dropped support for anything but Intel and Alpha. With SGI's decision to transition to the Itanium and IA32 architectures, use of MIPS processors on the desktop has now disappeared almost completely[1].

See main article Advanced Computing Environment.

Embedded markets

Through the 1990s, the MIPS architecture was widely adopted by the embedded market, including for use in computer networking/telecommunications, video arcade games, home video game consoles, computer printers, digital set-top boxes, digital televisions, DSL and cable modems, and personal digital assistants.

The low power-consumption and heat characteristics of embedded MIPS implementations, the wide availability of embedded development tools, and knowledge about the architecture means use of MIPS microprocessors in embedded roles is likely to remain common.

Synthesizeable Cores for Embedded Markets

In recent years most of the technology used in the various MIPS generations has been offered as IP-cores (building-blocks) for embedded processor designs. Both 32-bit and 64-bit basic cores are offered, known as the **4K** and **5K** respectively, and the design itself can be licensed as **MIPS32** and **MIPS64**. These cores can be mixed with add-in units such as FPUs, SIMD systems, various input/output devices, etc.

MIPS cores have been commercially successful, now being used in many consumer and industrial applications. MIPS cores can be found in newer Cisco, Linksys and Mikrotik's routerboard routers, cable modems and ADSL modems, smartcards, laser printer engines, set-top boxes, robots, handheld computers, Sony PlayStation 2 and Sony PlayStation Portable. In cellphone/PDA applications, the MIPS core has been unable to displace the incumbent, competing ARM core.

Examples of MIPS-powered devices: Broadcom BCM5352E - WiFi router processor with 54g WLAN, fast Ethernet, 200 MHz, 16KB ins. 8KB data cache, 256B prefetch cache, MMU, 16-bit 100 MHz SDRAM controller, serial/parallel flash, 5-port 100 Mbit/s Ethernet (switch), 16 GPIO, JTAG, 2xUART, 336-ball BGA. BCM 11xx, 12xx, 14xx - 64bit "SiByte" MIPS line.

MIPS architecture processors include: IDT RC32438; ATI Xilleon; Alchemy Au1000, 1100, 1200; Broadcom Sentry5; RMI XLR7xx, Cavium Octeon CN30xx, CN31xx, CN36xx, CN38xx and CN5xxx; Infineon Technologies EasyPort, Amazon, Danube, ADM5120, WildPass, INCA-IP, INCA-IP2; NEC EMMA and EMMA2, NEC VR4181A,

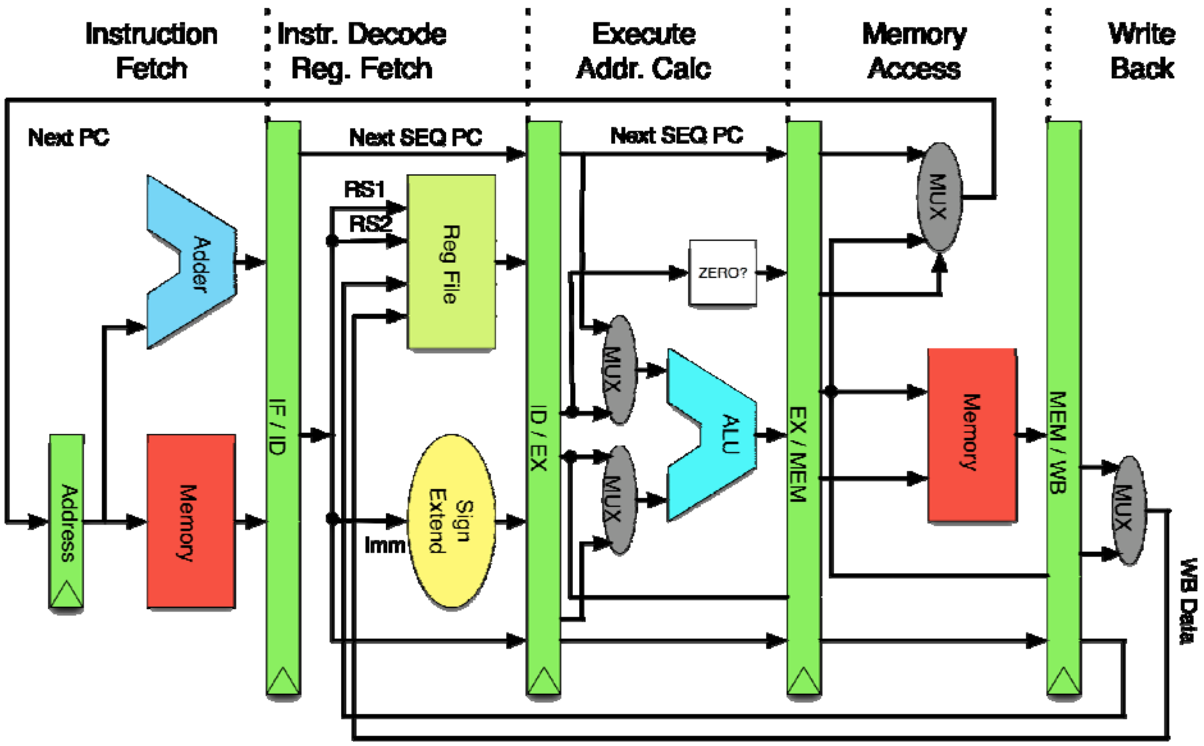
VR4121, VR4122, VR4181A, VR5432, VR5500; Oak Technologies Generation; PMC-Sierra RM11200; QuickLogic QuickMIPS ESP; Toshiba "Donau", Toshiba TMPR492x, TX4925, TX9956, TX7901.

MIPS based Supercomputers

One of the more interesting applications of the MIPS architecture is its use in massive processor count supercomputers. Silicon Graphics (SGI) refocused its business from desktop graphics workstations to the high performance computing (HPC) market in the early 1990s. The success of the company's first foray into server systems, the Challenge series based on the R4400 and R8000, and later **R10000**, motivated SGI to create a vastly more powerful system. The introduction of the integrated R10000 allowed SGI to produce a system, the Origin 2000, eventually scalable to 1024 CPUs using its NUMAlink cc-NUMA interconnect. The Origin 2000 begat the Origin 3000 series which topped out with the same 1024 maximum CPU count but using the R14000 and R16000 chips up to 700 MHz. Its MIPS based supercomputers were withdrawn in 2005 when SGI made the strategic decision to move to Intel's ***IA-64*** architecture.

An HPC startup introduced a radical MIPS based supercomputer in 2007. **SiCortex**, Inc. has created a tightly integrated Linux cluster supercomputer based on the MIPS64 architecture and a high performance interconnect based on the Kautz digraph topology. The system is very power efficient and computationally powerful. The most unique aspect of the system is its multicore processing node which integrates six MIPS64 cores, a crossbar memory controller, interconnect DMA engine, Gigabit Ethernet and PCI Express controllers all on a single chip which consumes only 10 watts of power, yet has a peak floating point performance of 6 GFLOPs. The most powerful configuration, the **SC5832**, is a single cabinet supercomputer consisting of **972** such node chips for a total of **5832** MIPS64 processor cores and **5.8** ***teraFLOPS*** of peak performance.

CPU family



Pipeline MIPS

The first commercial MIPS CPU model, the **R2000**, was announced in 1985. It added multiple-cycle multiply and divide instructions in a somewhat independent on-chip unit. New instructions were added to retrieve the results from this unit back to the execution core; these result-retrieving instructions were interlocked.

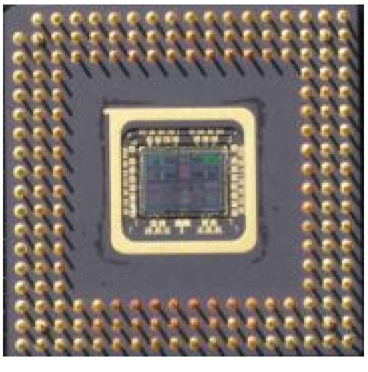
The R2000 could be booted either big-endian or little-endian. It had thirty-two 32-bit general purpose registers, but no condition code register (the designers considered it a potential bottleneck), a feature it shares with the AMD 29000 and the DEC Alpha. Unlike other registers, the program counter is not directly accessible.

The R2000 also had support for up to four co-processors, one of which was built into the main CPU and handled exceptions, traps and memory management, while the other three were left for other uses. One of these could be filled by the optional **R2010** FPU, which had thirty-two 32-bit registers that could be used as sixteen 64-bit registers for double-precision.

The **R3000** succeeded the R2000 in 1988, adding 32 KB (soon increased to 64 KB) caches for instructions and data, along with cache coherency support for multiprocessor use. While there were flaws in the R3000's multiprocessor support, it still managed to be a part of several successful multiprocessor designs. The R3000 also included a built-in MMU, a common feature on CPUs of the era. The R3000, like the R2000, could be paired with a **R3010** FPU. The R3000 was the first successful MIPS design in the marketplace, and eventually over one million were made. A speed-bumped version of the R3000 running up to 40 MHz, the **R3000A** delivered a performance of 32 VUPs (VAX Unit of Performance). The R3000A was the processor used in the extremely successful Sony PlayStation. Third-party designs include Performance Semiconductor's **R3400** and IDT's **R3500**, both of them were R3000As with an integrated R3010 FPU. Toshiba's **R3900** was a virtually first SoC for the early handheld PCs based on the Windows CE. A radiation-hardened variant for space applications, the Mongoose-V, is a R3000 with an integrated R3010 FPU.

The **R4000** series, released in 1991, extended the MIPS instruction set to a full 64-bit architecture, moved the FPU onto the main die to create a single-chip microprocessor, and operated at a radically high internal clock speed (it was introduced at 100 MHz). However, in order to achieve the clock speed the caches were reduced to 8 KB each and they took three cycles to access. The high operating frequencies were achieved through the technique of deep pipelining (called super-pipelining at the time). With the introduction of the R4000 a number of improved versions soon followed, including the **R4400** (1993) which included 16 KB caches, largely bug-free 64-bit operation, and support for a larger external level 2 cache.

MIPS, now a division of SGI called MTI, designed the lower-cost **R4200**, and later the even lower cost **R4300**, which was the R4200 with a 32-bit external bus. The Nintendo 64 used a NEC VR4300 CPU that was based upon the low-cost MIPS **R4300i**.[2]



bottom-side view of package of R4700 Orion with the exposed silicon chip, fabricated by IDT, designed by Quantum Effect Devices

topside view of package for R4700 Orion

Quantum Effect Devices (QED), a separate company started by former MIPS employees, designed the **R4600** "Orion", the **R4700** "Orion", the **R4650** and the **R5000**. Where the R4000 had pushed clock frequency and sacrificed cache capacity, the QED designs emphasized large caches which could be accessed in just two cycles and efficient use of silicon area. The R4600 and R4700 were used in low-cost versions of the SGI Indy workstation as well as the first MIPS based Cisco routers, such as the 36x0 and 7x00-series routers. The R4650 was used in the original WebTV set-top boxes (now Microsoft TV). The R5000 FPU had more flexible single precision floating-point scheduling than the R4000, and as a result, R5000-based SGI Indys had much better graphics performance than similarly clocked R4400 Indys with the same graphics hardware. SGI gave the old graphics board a new name when it was combined with R5000 in order to emphasize the improvement. QED later designed the **RM7000** and **RM9000** family of devices for embedded markets like networking and laser printers. QED was acquired by the semiconductor manufacturer PMC-Sierra in August 2000, the latter company continuing to invest in the MIPS architecture. The **RM7000** included an on-board 256 kB level 2 cache and a controller for optional level three cache. The **RM9xx0** were a family of SOC devices which included northbridge peripherals such as memory controller, PCI controller, gigabit ethernet controller and fast IO such as a hypertransport port.

The **R8000** (1994) was the first superscalar MIPS design, able to execute two integer or floating point and two memory instructions per cycle. The design was spread over six chips: an integer unit (with 16 KB instruction and 16 KB data caches), a floating-point unit, three full-custom secondary cache tag RAMs (two for secondary cache accesses, one for bus snooping), and a cache controller ASIC. The design had two fully pipelined double precision multiply-add units, which could stream data from the 4 MB off-chip secondary cache. The R8000 powered SGI's POWER Challenge servers in the mid 1990s and later became available in the POWER Indigo2 workstation. Although its FPU performance fit scientific users quite well, its limited integer performance and high cost dampened appeal for most users, and the R8000 was in the marketplace for only a year and remains fairly rare.

In 1995, the **R10000** was released. This processor was a single-chip design, ran at a faster clock speed than the R8000, and had larger 32 KB primary instruction and data caches. It was also superscalar, but its major innovation was out-of-order execution. Even with a single memory pipeline and simpler FPU, the vastly improved integer performance, lower price, and higher density made the R10000 preferable for most customers.

Recent designs have all been based upon R10000 core. The **R12000** used improved manufacturing to shrink the chip and operate at higher clock rates. The revised **R14000** allowed higher clock rates with additional support for DDR SRAM in the off-chip cache, and a faster front side bus clocked to 200 MHz for better throughput. Later iterations are named the **R16000** and the **R16000A** and feature increased clock speed, additional L1 cache, and smaller die manufacturing compared with before.

Other members of the MIPS family include the **R6000**, an ECL implementation of the MIPS architecture which was produced by Bipolar Integrated Technology. The R6000 microprocessor introduced the MIPS II instruction set. Its TLB and cache architecture are different from all other members of the MIPS family. The R6000 did not deliver the promised performance benefits, and although it saw some use in Control Data machines, it quickly disappeared from the mainstream market.

MIPS Microprocessors

Model	Frequency (MHz)	Year	Process (µm)	Transistors (Millions)	Die Size (mm²)	Pin Count	Power (W)	Voltage	Dcache (KB)	Icache (KB)	L2 Cache	L3 Cache
R2000	8-16.67	1985	2.0	0.11	?	?	?	?	32	64	None	None
R3000	12-40	1988	1.2	0.11	66.12	145	4	?	64	64	0-256 KB External	None
R4000	100	1991	0.8	1.35	213	179	15	5	8	8	1 MB External	None
R4400	100-250	1992	0.6	2.3	186	179	15	5	16	16	1-4 MB External	None
R4600	100-133	1994	0.64	2.2	77	179	4.6	5	16	16	512 KB External	None
R5000	150-200	1996	0.35	3.7	84	223	10	3.3	32	32	1 MB External	None
R8000	75-90	1994	0.7	2.6	299	591+591	30	3.3	16	16	4 MB External	None
R10000	150-250	1996	0.35, 0.25	6.7	299	599	30	3.3	32	32	1-4 MB External	None
R12000	270-400	1998	0.25, 0.18	6.9	204	600	20	4	32	32	2 MB External	None
RM7000	250-600	1998	0.25, 0.18, 0.13	18	91	304	10, 6, 3	3.3, 2.5, 1.5	16	16	256 KB Internal	1 MB External
R14000	500-600	2001	0.13	7.2	204	527	17	?	32	32	2-4 MB External	None
R16000	700-1000	2002	0.11	?	?	?	20	?	64	64	4-16 MB External	None

Note: These specifications are for common processor models. Variations exist, especially in Level 2 cache.

Note: The R8000 has a unique cache hierarchy named 'Data Streaming Cache' where there is 16 KB of L1 data cache for the integer chip with an external 4 MB L2 cache that served as the secondary unified cache for the integer chip but as the L1 data cache for the floating point chip.

Summary of R3000 instruction set Opcodes

Instructions are divided into three types: R, I and J. Every instruction starts with a 6-bit opcode. In addition to the opcode, R-type instructions specify three registers, a shift amount field, and a function field; I-type instructions specify two registers and a 16-bit immediate value; J-type instructions follow the opcode with a 26-bit jump target.[3][4]

The following are the three formats used for the core instruction set:

Type	-31-	format (bits)				-0-
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

MIPS Assembly Language

These are assembly language instructions that have direct hardware implementation, as opposed to *pseudoinstructions* which are translated into multiple real instructions before being assembled.

- **CONST denotes a constant ("immediate").**
- **In the following, the register numbers are only examples, and any other registers can be used in their places.**
- **All the following instructions are native instructions.**
- **Opcodes and funct codes are in hexadecimal.**
- **The MIPS32 Instruction Set** states that the word **unsigned** as part of Add and Subtract instructions, is a *misnomer*. The difference between *signed* and *unsigned* versions of commands is not a sign extension (or lack thereof) of the operands, but controls whether a trap is executed on overflow *(e.g. Add)* or an overflow is ignored *(Add unsigned)*. An immediate operand CONST to these instructions is always sign-extended.

Category	Name	Instruction syntax	Meaning	Format/opcode/funct			Notes
Arithmetic	Add	add $1,$2,$3	$1 = $2 + $3	R	0	20_{16}	adds two registers, executes a trap on overflow
	Add unsigned	addu $1,$2,$3	$1 = $2 + $3	R	0	21_{16}	as above but ignores an overflow
	Subtract	sub $1,$2,$3	$1 = $2 - $3	R	0	22_{16}	subtracts two registers, executes a trap on overflow
	Subtract unsigned	subu $1,$2,$3	$1 = $2 - $3	R	0	23_{16}	as above but ignores an overflow
	Add immediate	addi $1,$2,CONST	$1 = $2 + CONST (signed)	I	8_{16}		Used to add sign-extended constants (and also to copy one register to another "addi $1, $2, 0"), executes a trap on overflow
	Add immediate unsigned	addiu $1,$2,CONST	$1 = $2 + CONST (signed)	I	9_{16}		as above but ignores an overflow, CONST still sign-extended
	Multiply	mult $1,$2	LO = (($1 * $2) << 32) >> 32; HI = ($1 * $2) >> 32;	R	0	18_{16}	Multiplies two registers and puts

							the 64-bit result in two special memory spots - LOW and HI. Alternatively, one could say the result of this operation is: (int HI,int LO) = (64-bit) $1 * $2 . See mfhi and mflo for accessing LO and HI regs.
	Divide	div $1, $2	LO = $1 / $2 HI = $1 % $2	R			Divides two registers and puts the 32-bit integer result in LO and the remainder in HI.[3]
Data Transfer	Load double word	ld $1,CONST($2)	$1 = Memory[$2 + CONST]	I	23_{16}		loads the word stored from: MEM[$2+CONST] and the following 7 bytes to $1 and the next register.
	Load word	lw $1,CONST($2)	$1 = Memory[$2 + CONST]	I	23_{16}		loads the word stored from: MEM[$2+CONST] and the following 3 bytes.
	Load halfword	lh $1,CONST($2)	$1 = Memory[$2 + CONST] (signed)	I	25_{16}		loads the halfword stored from: MEM[$2+CONST] and the following byte. Sign is extended to width of register.
	Load halfword unsigned	lhu $1,CONST($2)	$1 = Memory[$2 + CONST] (unsigned)	I			As above without sign extension.
	Load byte	lb $1,CONST($2)	$1 = Memory[$2 + CONST] (signed)	I			loads the byte stored from: MEM[$2+CONST].
	Load byte unsigned	lbu $1,CONST($2)	$1 = Memory[$2 + CONST] (unsigned)	I			As above without sign extension.
	Store double word	sd $1,CONST($2)	Memory[$2 + CONST] = $1	I			stores two words from $1 and the next register into: MEM[$2+CONST] and the following 7 bytes. The order of the operands is a

							large source of confusion.
	Store word	sw $1,CONST($2)	Memory[$2 + CONST] = $1	I			stores a word into: MEM[$2+CONST] and the following 3 bytes. The order of the operands is a large source of confusion.
	Store half	sh $1,CONST($2)	Memory[$2 + CONST] = $1	I			stores the first half of a register (a halfword) into: MEM[$2+CONST] and the following byte.
	Store byte	sb $1,CONST($2)	Memory[$2 + CONST] = $1	I			stores the first fourth of a register (a byte) into: MEM[$2+CONST].
	Load upper immediate	lui $1,CONST	$1 = CONST << 16	I			loads a 16-bit immediate operand into the upper 16-bits of the register specified. Maximum value of constant is 2^{16}-1
	Move from high	mfhi $1	$1 = HI	R			Moves a value from HI to a register. Do not use a multiply or a divide instruction within two instructions of mfhi (that action is undefined because of the MIPS pipeline).
	Move from low	mflo $1	$1 = LO	R	0	12_{16}	Moves a value from LO to a register. Do not use a multiply or a divide instruction within two instructions of mflo (that action is undefined because of the MIPS pipeline).
	Move from Control	mfcZ $1, $2	$1 = Coprocessor[Z].ControlRegister[$2]	R			Moves a 4 byte value from Coprocessor Z

	Register						Control register to a general purpose register. Sign extension.
	Move to Control Register	mtcZ $1, $2	Coprocessor[Z].ControlRegister[$2] = $1	R			Moves a 4 byte value from a general purpose register to a Coprocessor Z Control register. Sign extension.
	Load word coprocessor	lwcZ $1,CONST ($2)	Coprocessor[Z].DataRegister[$1] = Memory[$2 + CONST]	I			Loads the 4 byte word stored from: MEM[$2+CONST] into a Coprocessor data register. Sign extension.
	Store word coprocessor	swcZ $1,CONST ($2)	Memory[$2 + CONST] = Coprocessor[Z].DataRegister[$1]	I			Stores the 4 byte word held by a Coprocessor data register into: MEM[$2+CONST]. Sign extension.
Logical	And	and $1,$2,$3	$1 = $2 & $3	R			Bitwise and
	And immediate	andi $1,$2,CONST	$1 = $2 & CONST	I			
	Or	or $1,$2,$3	$1 = $2 | $3	R			Bitwise or
	Or immediate	ori $1,$2,CONST	$1 = $2 | CONST	I			
	Exclusive or	xor $1,$2,$3	$1 = $2 ^ $3	R			
	Nor	nor $1,$2,$3	$1 = ~ ($2 | $3)	R			Bitwise nor
	Set on less than	slt $1,$2,$3	$1 = ($2 < $3)	R			Tests if one register is less than another.
	Set on less than immediate	slti $1,$2,CONST	$1 = ($2 < CONST)	I			Tests if one register is less than a constant.
Bitwise Shift	Shift left logical	sll $1,$2,CONST	$1 = $2 << CONST	R			shifts CONST number of bits to the left (multiplies by 2^{CONST})
	Shift right logical	srl $1,$2,CONST	$1 = $2 >> CONST	R			shifts CONST number of bits to the right - zeros are shifted in (divides by 2^{CONST}). Note that this instruction only works as division of a two's

							complement number if the value is positive.
	Shift right arithmetic	sra $1,$2,CONST	$\$1 = \$2 >> CONST + \left(\left(\sum_{n=1}^{CONST} 2^{31-n}\right) \cdot \$2 >> 31\right)$	R			shifts CONST number of bits - the sign bit is shifted in (divides 2's complement number by 2^{CONST})
Conditional branch	Branch on equal	beq $1,$2,CONST	if ($1 == $2) go to PC+4*CONST	I			Goes to the instruction at the specified address if two registers are equal.
	Branch on not equal	bne $1,$2,CONST	if ($1 != $2) go to PC+4*CONST	I			Goes to the instruction at the specified address if two registers are *not* equal.
Unconditional jump	Jump	j CONST	goto address CONST	J			Unconditionally jumps to the instruction at the specified address.
	Jump register	jr $1	goto address $1	R			Jumps to the address contained in the specified register
	Jump and link	jal CONST	$31 = PC + 4; goto CONST	J			For procedure call - used to call a subroutine, $31 holds the return address; returning from a subroutine is done by: jr $31

NOTE: in the branching and jump instructions, the offset can be replaced by a label present somewhere in the code.

NOTE: that there is no corresponding "load lower immediate" instruction; this can be done by using addi (add immediate, see below) or ori (or immediate) with the register $0 (whose value is always zero). For example, both addi $1, $0, 100 and ori $1, $0, 100 load the decimal value 100 into register $1.

NOTE: An arithmetic operation with signed immediates differs from one with unsigned ones in that it does not throw an exception. Subtracting an immediate can be done with adding the negation of that value as the immediate.

Pseudo instructions

These instructions are accepted by the MIPS assembler, however they are not real instructions within the MIPS instruction set. Instead, the assembler translates them into sequences of real instructions.

Name	**instruction syntax**	**Real instruction translation**	**meaning**
Load Address	la $1, LabelAddr	lui $1, LabelAddr[31:16]; ori $1,$1,	$1 = Label Address

		LabelAddr[15:0]	
Load Immediate	li $1, IMMED[31:0]	lui $1, IMMED[31:16]; ori $1,$1, IMMED[15:0]	$1 = 32 bit Immediate value
Branch if greater than	bgt		if(R[rs]>R[rt]) PC=Label
Branch if less than	blt		if(R[rs]<R[rt]) PC=Label
Branch if greater than or equal	bge		if(R[rs]>=R[rt]) PC=Label
branch if less than or equal	ble		if(R[rs]<=R[rt]) PC=Label
branch if greater than unsigned	bgtu		if(R[rs]=>R[rt]) PC=Label
branch if greater than zero	bgtz		if(R[rs]>0) PC=Label

Some other important instructions

- nop (no operation) (machine code 0x00000000, interpreted by CPU as sll $0,$0,0)
- break (breaks the program, used by debuggers)
- syscall (used for system calls to the operating system)
- **a full set of Floating point instructions for both single precision and double precision operands**

Compiler Register Usage

Main article: calling convention#MIPS

The hardware architecture specifies that:

- General purpose register $0 always returns a value of 0 .
- General purpose register $31 is used as the link register for jump and link instructions.
- HI and LO are used to access the multiplier/divider results, accessed by the mfhi (move from high) and mflo commands.

These are the only hardware restrictions on the usage of the general purpose registers.

The various MIPS tool-chains implement specific calling conventions that further restrict how the registers are used. These calling conventions are totally maintained by the tool-chain software and are not required by the hardware.

Registers

Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	no
$v0–$v1	$2–$3	Values for function returns and expression evaluation	no
$a0–$a3	$4–$7	function arguments	no
$t0–$t7	$8–$15	temporaries	no
$s0–$s7	$16–$23	saved temporaries	**yes**
$t8–$t9	$24–$25	temporaries	no
$k0–$k1	$26–$27	reserved for OS kernel	no

$gp	$28	global pointer	**yes**
$sp	$29	stack pointer	**yes**
$fp	$30	frame pointer	**yes**
$ra	$31	return address	N/A

Registers that are preserved across a call are registers that (by convention) will not be changed by a system call or procedure (function) call. For example, $s-registers must be saved to the stack by a procedure that needs to use them, and $sp and $fp are always incremented by constants, and decremented back after the procedure is done with them (and the memory they point to). By contrast, $ra is changed automatically by any normal function call (ones that use jal), and $t-registers must be saved by the program before any procedure call (if the program needs the values inside them after the call).

Simulators

Open Virtual Platforms (OVP) [1] includes the freely available simulator OVPsim, a library of models of processors, peripherals and platforms, and APIs which enable users to develop their own models. The models in the library are open source, written in C, and include the MIPS 4K, 24K and 34K cores. These models are created and maintained by Imperas [2] and in partnership with MIPS Technologies have been tested and assigned the MIPS-Verified(tm) mark. The OVP site also includes models of ARM, Tensilica and OpenCores/openRisc processors. Sample MIPS-based platforms include both bare metal environments and platforms for booting unmodified Linux binary images. These platforms/emulators are available as source or binaries and are fast, free, and easy to use. OVPsim is developed and maintained by Imperas and is very fast (100s of million instructions per second), and built to handle multicore architectures. To download the MIPS OVPsim simulators/emulators visit [3].

There is a freely available "MIPS32 Simulator" (earlier versions simulated only the R2000/R3000) called SPIM for several operating systems (specifically Unix or GNU/Linux; Mac OS X; MS Windows 95, 98, NT, 2000, XP; and DOS) which is good for learning MIPS assembly language programming and the general concepts of RISC-assembly language programming: http://www.cs.wisc.edu/~larus/spim.html

EduMIPS64 is a GPL graphical cross-platform MIPS64 CPU simulator, written in Java/Swing. It supports a wide subset of the MIPS64 ISA and allows the user to graphically see what happens in the pipeline when an assembly program is run by the CPU. It has educational purposes and is used in some Computer Architecture courses in Universities around the world. More info at http://www.edumips.org

MARS is another GUI based MIPS emulator designed for use in education, specifically for use with Hennessy's Computer Organization and Design. More information is available at http://courses.missouristate.edu/KenVollmar/MARS/

More advanced free MIPS emulators are available from the GXemul (formerly known as the mips64emul project) and QEMU projects, which emulate not only the various MIPS III and higher microprocessors (from the R4000 through the R10000), but also entire computer systems which use the microprocessors. For example, GXemul can emulate both a DECstation with a MIPS R4400 CPU (and boot to Ultrix), and an SGI O2 with a MIPS R10000 CPU (although the ability to boot Irix is limited), among others, as well as the various framebuffers, SCSI controllers, and the like which comprise those systems.

Commercial simulators are available especially for the embedded use of MIPS processors, for example Virtutech Simics (MIPS 4Kc and 5Kc, PMC RM9000, QED RM7000), VaST Systems (R3000, R4000), and CoWare (the MIPS4KE, MIPS24K, MIPS25Kf and MIPS34K).

Examples of system calls (used by SPIM)

service	**Trap code**	**Input**	**Output**	**Notes**
print_int	$v0 = 1	$a0 = integer to print	prints $a0 to standard output	
print_float	$v0 = 2	$f12 = float to print	prints $f12 to standard output	
print_double	$v0 = 3	$f12 = double to print	prints $f12 to standard output	
print_string	$v0 = 4	$a0 = address of first character		prints a character string to standard output
read_int	$v0 = 5		integer read from standard input placed in $v0	
read_float	$v0 = 6		float read from standard input placed in $f0	
read_double	$v0 = 7		double read from standard input placed in $f0	
read_string	$v0 = 8	$a0 = address to place string, $a1 = max string length	reads standard input into address in $a0	
sbrk	$v0 = 9	$a0 = number of bytes required	$v0= address of allocated memory	Allocates memory from the heap
exit	$v0 = 10			
print_char	$v0 = 11	$a0 = character (low 8 bits)		
read_char	$v0 = 12		$v0 = character (no line feed) echoed	
file_open	$v0 = 13	$a0 = full path (zero terminated string with no line feed), $a1 = flags, $a2 = UNIX octal file mode (0644 for rw-r--r--)	$v0 = file descriptor	
file_read	$v0 = 14	$a0 = file descriptor, $a1 = buffer address, $a2 = amount to read in bytes	$v0 = amount of data in buffer from file (-1 = error, 0 = end of file)	
file_write	$v0 = 15	$a0 = file descriptor, $a1 = buffer address, $a2 = amount to write in bytes	$v0 = amount of data in buffer to file (-1 = error, 0 = end of file)	
file_close	$v0 = 16	$a0 = file descriptor		

Flags:

Read = 0x0, Write = 0x1, Read/Write = 0x2

OR Create = 0x100, Truncate = 0x200, Append = 0x8

OR Text = 0x4000, Binary = 0x8000

Trivia

- The rabbit in Super Mario 64 is named MIPS after the technology because the Nintendo 64 used it.

Notes

1. ^ SGI announcing the end of MIPS
2. ^ NEC Offers Two High Cost Performance 64-bit RISC Microprocessors
3. ^ [a] [b] MIPS R3000 Instruction Set Summary
4. ^ MIPS Instruction Reference

Further reading

- Patterson, David A; John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers. ISBN 1-55860-604-1.
- Sweetman, Dominic. *See MIPS Run*. Morgan Kaufmann Publishers. ISBN 1-55860-410-3.
- Farquhar, Erin; Philip Bunce. *MIPS Programmer's Handbook*. Morgan Kaufmann Publishers. ISBN 1-55860-297-6.

See also

- DLX, a very similar architecture designed by John L. Hennessy (creator of MIPS) for teaching purposes
- Loongson, a MIPS-like processor architecture developed at Chinese Academy of Sciences
- MIPS-X, developed as a follow-on project to the MIPS architecture
- Mongoose-V, a radiation hardened version of the MIPS R3000 used in spacecrafts

External links

Wikibooks has a book on the topic of
MIPS Assembly

- Full overview of MIPS architecture.
- Patterson & Hennessy - Appendix A (PDF)
- summary of MIPS assembly language
- MIPS Instruction reference
- MIPS processor images and descriptions at cpu-collection.de
- A programmed introduction to MIPS assembly
- mips bitshift operators
- MIPS software user's manual