MIPS?

MIPS (**Microprocessor without Interlocked Pipelined Stages**) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems, now MIPS Technologies, based in the United States.

There are multiple versions of MIPS: including MIPS I, II, III, IV, and V; as well as five releases of MIPS32/64 (for 32- and 64-bit implementations, respectively). The early MIPS architectures were 32-bit only; 64-bit versions were developed later. As of April 2017, the current version of MIPS is MIPS32/64 Release 6. MIPS32/64 primarily differs from MIPS I–V by defining the privileged kernel mode System Control Coprocessor in addition to the user mode architecture.

The MIPS architecture has several optional extensions. 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, and MIPS MT, which adds multithreading capability.

Computer architecture courses in universities and technical schools often study the MIPS architecture. The architecture greatly influenced later RISC architectures such as Alpha. In March 2021, MIPS announced that the development of the MIPS architecture had ended as the company is making the transition to RISC-V.

History??

The first version of the MIPS architecture was designed by MIPS Computer Systems for its R2000 microprocessor, the first MIPS implementation. Both MIPS and the R2000 were introduced together in 1985. When MIPS II was introduced, *MIPS* was renamed *MIPS I* to distinguish it from the new version.

MIPS Computer Systems' R6000 microprocessor (1989) was the first MIPS II implementation. Designed for servers, the R6000 was fabricated and sold by Bipolar Integrated Technology, but was a commercial failure. During the mid-1990s, many new 32-bit MIPS processors for embedded systems were MIPS II implementations because the introduction of the 64-bit MIPS III architecture in 1991 left MIPS II as the newest 32-bit MIPS architecture until MIPS32 was introduced in 1999.

MIPS Computer Systems' R4000 microprocessor (1991) was the first MIPS III implementation. It was designed for use in personal, workstation, and server computers. MIPS Computer Systems aggressively promoted the MIPS architecture and R4000, establishing the Advanced Computing Environment (ACE) consortium to advance its Advanced RISC Computing (ARC) standard, which aimed to establish MIPS as the dominant personal computing platform. ARC found little success in personal computers, but the R4000 (and the R4400 derivative) were widely used in workstation and server computers, especially by its largest user, Silicon Graphics. Other uses of the R4000 included high-end embedded systems and supercomputers. MIPS III was eventually implemented by a number of embedded

microprocessors. Quantum Effect Design's R4600 (1993) and its derivatives was widely used in high-end embedded systems and low-end workstations and servers. MIPS Technologies' R4200 (1994), was designed for embedded systems, laptop, and personal computers. A derivative, the R4300i, fabricated by NEC Electronics, was used in the Nintendo 64 game console. The Nintendo 64, along with the PlayStation, were among the highest volume users of MIPS architecture processors in the mid-1990s.

The first MIPS IV implementation was the MIPS Technologies R8000 microprocessor chipset (1994). The design of the R8000 began at Silicon Graphics, Inc. and it was only used in high-end workstations and servers for scientific and technical applications where high performance on large floating-point workloads was important. Later implementations were the MIPS Technologies R10000 (1996) and the Quantum Effect Devices R5000 (1996) and RM7000 (1998). The R10000, fabricated and sold by NEC Electronics and Toshiba, and its derivatives were used by NEC, Pyramid Technology, Silicon Graphics, Inc., and Tandem Computers (among others) in workstations, servers, and supercomputers. The R5000 and R7000 found use in high-end embedded systems, personal computers, and low-end workstations and servers. A derivative of the R5000 from Toshiba, the R5900, was used in Sony Computer Entertainment's Emotion Engine, which powered its PlayStation 2 game console.

Announced on October 21, 1996 at the Microprocessor Forum 1996 alongside the MIPS Digital Media Extensions (MDMX) extension, MIPS V was designed to improve the performance of 3D graphics transformations. In the mid-1990s, a major use of non-embedded MIPS microprocessors were graphics workstations from SGI. MIPS V was completed by the integer-only MDMX extension to provide a complete system for improving the performance of 3D graphics applications. MIPS V implementations were never introduced. On May 12, 1997, SGI announced the "H1" ("Beast") and "H2" ("Capitan") microprocessors. The former was to have been the first MIPS V implementation, and was due to be introduced in the first half of 1999. The "H1" and "H2" projects were later combined and were eventually cancelled in 1998. While there have not been any MIPS V implementations, MIPS64 Release 1 (1999) was based on MIPS V and retains all of its features as an optional Coprocessor 1 (FPU) feature called Paired-Single.

When MIPS Technologies was spun-out of Silicon Graphics in 1998, it refocused on the embedded market. Up to MIPS V, each successive version was a strict superset of the previous version, but this property was found to be a problem, and the architecture definition was changed to define a 32-bit and a 64-bit architecture: MIPS32 and MIPS64. Both were introduced in 1999. MIPS32 is based on MIPS II with some additional features from MIPS III, MIPS IV, and MIPS V; MIPS64 is based on MIPS V. NEC, Toshiba and SiByte (later acquired by Broadcom) each obtained licenses for MIPS64 as soon as it was announced. Philips, LSI Logic, IDT, Raza Microelectronics, Inc., Cavium, Loongson Technology and Ingenic Semiconductor have since joined them. MIPS32/MIPS64 Release 5 was announced on December 6, 2012. Release 4 was skipped because the number four is perceived as unlucky in many Asian cultures.

In December 2018, Wave Computing, the new owner of the MIPS architecture, announced that MIPS ISA would be open-sourced in a program dubbed the MIPS Open initiative. The

program was intended to open up access to the most recent versions of both the 32-bit and 64-bit designs making them available without any licensing or royalty fees as well as granting participants licenses to existing MIPS patents.

In March 2019, one version of the architecture was made available under a royalty-free license, but later that year the program was shut down again.

In March 2021, Wave Computing announced that the development of the MIPS architecture has ceased. The company has joined the RISC-V foundation and future processor designs will be based on the RISC-V architecture.

Design?

MIPS is a modular architecture supporting up to four coprocessors (CP0/1/2/3). In MIPS terminology, CP0 is the System Control Coprocessor (an essential part of the processor that is implementation-defined in MIPS I–V), CP1 is an optional floating-point unit (FPU) and CP2/3 are optional implementation-defined coprocessors (MIPS III removed CP3 and reused its opcodes for other purposes). For example, in the PlayStation video game console, CP2 is the Geometry Transformation Engine (GTE), which accelerates the processing of geometry in 3D computer graphics.

Version?

MIPS32/MIPS64?

The first release of MIPS32, based on MIPS II, added conditional moves, prefetch instructions, and other features from the R4000 and R5000 families of 64-bit processors. The first release of MIPS64 adds a MIPS32 mode to run 32-bit code. The MUL and MADD (multiply-add) instructions, previously available in some implementations, were added to the MIPS32 and MIPS64 specifications, as were cache control instructions.

MIPS32/MIPS64 Release 6 in 2014 added the following:

- a new family of branches with no delay slot:
    - unconditional branches (BC) and branch-and-link (BALC) with a 26-bit offset,
    - conditional branch on zero/non-zero with a 21-bit offset,
    - full set of signed and unsigned conditional branches compare between two registers (e.g. BGTUC) or a register against zero (e.g. BGTZC),
    - full set of branch-and-link which compare a register against zero (e.g. BGTZALC).
- index jump instructions with no delay slot designed to support large absolute addresses.
- instructions to load 16-bit immediates at bit position 16, 32 or 48, allowing to easily generate large constants.
- PC-relative load instructions, as well as address generation with large (PC-relative) offsets.

- bit-reversal and byte-alignment instructions (previously only available with the DSP extension).
- multiply and divide instructions redefined so that they use a single register for their result).
- instructions generating truth values now generate all zeroes or all ones instead of just clearing/setting the 0-bit,
- instructions using a truth value now only interpret all-zeroes as false instead of just looking at the 0-bit.

Removed infrequently used instructions:

- some conditional moves
- *branch likely* instructions (deprecated in previous releases).
- integer overflow trapping instructions with 16-bit immediate
- integer accumulator instructions (together HI/LO registers, moved to the DSP Application-Specific Extension)
- unaligned load instructions (LWL and LWR), (requiring that most ordinary loads and stores support misaligned access, possibly via trapping and with the addition of a new instruction (BALIGN))

Reorganized the instruction encoding, freeing space for future expansions.

Application Specific Extension

MIPS SIMD architecture (MSA) [SISD, SIMD, MISD-Failure, MIMD-IBM T J WATSON Supercomputer]

Instruction set extensions designed to accelerate multimedia.

- 32 vector registers of 16 x 8-bit, 8 x 16-bit, 4 x 32-bit, and 2 x 64 bit vector elements
- Efficient vector parallel arithmetic operations on integer, fixed-point and floating-point data
- Operations on absolute value operands
- Rounding and saturation options available
- Full precision multiply and multiply-add
- Conversions between integer, floating-point, and fixed-point data
- Complete set of vector-level compare and branch instructions with no condition flag
- Vector (1D) and array (2D) shuffle operations
- Typed load and store instructions for endian-independent operation
- IEEE Standard for Floating-Point Arithmetic 754-2008 compliant
- Element precise floating-point exception signaling
- Pre-defined scalable extensions for chips with more gates/transistors
- Accelerates compute-intensive applications in conjunction with leveraging generic compiler support
- Software-programmable solution for consumer electronics applications or functions not covered by dedicated hardware
- Emerging data mining, feature extraction, image and video processing, and human-computer interaction applications

- High-performance scientific computing

Important Points

1. Since, the MIPS doesn't have any Flags(It uses Internal Registers to represent flag values), there will be no flag overlapping while pipelining.
2. Assume, during pipelining , instructions get overlapped and hence there will be a chance to edit the pre-modified flag. But MIPS overcomes this problems, with internal registers instead of separate flag registers.

What is MIPS Architecture?

MIPS is a RISC (reduced instruction set computing) instruction set architecture developed by several Stanford researchers in the mid 1980s. Originally, the name was an acronym for Microprocessor without Interlocked Pipeline Stages, but interlocks between pipeline stages were eventually reintroduced, probably for performance reasons as other processors became more advanced. The decision for making a processor without interlocking pipeline stages was based on performance and simplicity of design. With interlocks, operations such as integer division, which is very time-consuming, would cause other pipeline phases to wait until the execute unit was done with the division. This defeats the purpose of pipelining because it causes sections of the processor to idle. Reducing all phases to one clock cycle removes idling (however it might force the clock to be slower).

MIPS follows the classic 5-stage RISC pipeline: Instruction Fetch (IF), Instruction Decode/Operand Fetch (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). MIPS is a load-store architecture, which means that to do arithmetic on data, values must explicitly be read from memory with a special load instruction and written to memory with a store instruction; arithmetic instructions only operate on registers.

In my opinion, MIPS is a really great architecture if you're interested in learning about ISAs, computer architecture, and computer organization because it is straightforward and simple, yet doesn't do away with important functionality.

What is the actual difference between x86, ARM and MIPS architectures?

The actual differences between the three are too many for an answer here. Most of these subtle differences lie in the way ***memory is addressed, exceptions are handled, branches are executed etc.*** There are many subtle differences too that are beyond the scope of a brief answer here. From a high level though, I can think of these:

1. RISC vs CISC: This is the classic difference between ARM/MIPS and x86. RISC stands for Reduced Instruction Set Computer and CISC is Complete ISC. x86 is a CISC processor and both ARM/MIPS are RISC. The philosophy behind CISC processors is that a single instruction can do multiple things; like add an immediate number with a register, store this value to an address computed using some other register and also set arithmetic flags. RISC processors on the other hand would have two separate instructions for this; one to add two numbers and the other to store the result. This increases the

instruction count but makes the instructions simpler. Which brings me to the second point.

2. Power vs Performance: Simpler instructions tend to consume lesser power. x86 processors are typically designed for high performance applications like servers, while RISC processors are used in mobile applications where performance is sometimes compromised for better power efficiency. It should be noted however that these lines are thinning down every day. ARM has added more complexity to its instructions to get better performance and Intel breaks down its opcodes into micro-ops that are ARM like to achieve better power. MIPS perhaps remains the simplest ISA around and is used more heavily in embedded applications where ultra-low power consumption is needed.
3. Security and virtualization: Not sure about MIPS here, but ARM has something called TrustZone and Intel has vPro. Both these features target hardware security. Both ARM and Intel have support for virtualization and memory management, and they differ only in their implementation. These are rather new features that are needed in newer applications and thus, both ISAs have one or the other flavor. So you see, the differences are trimming down.

All in all, the ISAs matter only from the programming point at this time. The mobile software ecosystem is way better developed and maintained for ARM than it is for x86. The picture reverses when it comes to desktop applications. That is the main reason why Intel is struggling to get into the mobile market and ARM is having a hard time to get into servers. In all of this, MIPS is there somewhere along with Sparc, PowerPC etc being used in rather niche applications.

Talking with respect to their **ISA** (Instruction Set Architecture), actual differences between their architectures are:

1. ***Class of ISA***: x86 architecture has a register-memory ISA where many instructions can access the memory directly. ***MIPS and ARM have load-store ISAs wherein only load and store instructions can access the memory.***
2. ***Memory addressing:*** All the 3 architectures use byte addressing to access memory operands. ***MIPS and ARM architecture require the objects to be aligned which makes accesses faster but object alignment is not a compulsion in x86 architecture.***
3. ***Addressing modes: MIPS has 3 addressing modes: register, immediate and displacement.*** x86 has in total 6 addressing modes: 3 modes supported by MIPS plus 3 different variants of displacement type addressing. ARM has 3 MIPS addressing modes plus PC-relative addressing, the sum of two registers mode and the sum of two registers where one register is multiplied by the size of the operand in bytes. It also has 2 unique addressing modes: auto-increment and auto-decrement addressing modes.
4. ***Types of sizes and operands:*** All 3 of them support operand sizes of 8-bit (ASCII character), 16-bit (Unicode character or half word), 32-bit (integer or word), 64-bit (double word or long integer), and floating point in 32-bit (single precision) and 64-bit (double precision). The x86 architecture also supports 80-bit floating point (extended double precision).

5. ***Control flow instructions:*** All 3 of them use PC-relative addressing where the branch address is specified and support conditional branches, unconditional jumps, procedure calls, and returns. ***MIPS conditional branches (BE, BNE, etc.) test the contents of registers, while the x86 and ARM branches test condition code bits set as side after performing arithmetic and logic operations.*** The ARM and MIPS procedure call places the return address in a register, while the x86 call (CALLF) places the return address on a stack in memory.
6. ***Encoding an ISA:*** There are 2 types of encoding: fixed length and variable length. ***ARM and MIPS instructions are encoded using fixed length encoding as their instructions are 32-bits long.*** The x86 encoding is variable length, ranging from 1 to 18 bytes. Variable length instructions can take less space than fixed-length instructions, so a program compiled for the x86 is usually smaller than the same program compiled for MIPS. ARM and MIPS later offered extensions to offer 16-bit length instructions so as to reduce program size, called Thumb or Thumb-2 and MIPS16, respectively. Number of registers and addressing modes both have a significant impact on the size of instructions as both these fields appear several times in a single instruction.

More Knowledge

Here I'm really comparing the instruction sets, not the average processors that implement them.

X86 is based on CISC philosophy.

In x86, most instructions can take one of their arguments directly from the memory, without needing a separate load instructions. Because of this, it needs (and has) less registers than most other architectures. Memory operations can have quite complex addressing modes(saving instructions wasted on address calculations). Instructions are also variable-length, the most-common-used instruction are shorter. The variable-length instruction encoding helps x86 to achieve smaller code size (which helps to have good instruction cache hit rates, and program binaries that are smaller), but makes instruction decoding more complex. The more complex instruction decoding adds 1–2 pipeline stages and limits the decoding throughput to about 4 instructions/cycle. Because of this, most new high-bandwidth x86 CPUs have an L0 instruction cache which stores the already-decoded micro-ops.

MIPS is based on pure 1980's RISC philosophy.

It was designed to fit very nicely with the 1980s 5-stage RISC pipeline model. All operands that come from the memory have to be loaded from the memory with separate load instructions before they can be used. It has delay slots: The instruction after a branch is always executed even if the branch is taken. It also has some of it's own strange things, like special hi and lo registers for multiplication and division. Memory addressing modes are very limited, only base+offset where offset is an immediate(constant). This prevent for example code where an

array is indexed and an value form array is loaded with single instruction in a loop; This requires three instructions in MIPS.

MIPS is simply trying to solve the wrong problem, with solutions that are fixed to 1980s; It's adding all kind of strange things just to make instruction set easily pipelineable with this kind of pipeline. But none of these strange things are really needed to make a pipelined processor, and many of these actually make it HARDER to make longer pipelines, they are mostly only good for five for ~5 pipeline stages of 1980s. For example, instead of ONE branch delay slots, modern processors would need about 60 delay slots to achieve the full benefit from them. With modern pipeline lengths, one delay slot is not helping practically anything, it's just making things more complex.

In normal MIPS instruction set, all instructions are 32-bit long, and many instructions are typically needed to get anything done, so MIPS is quite bad for code density. Program binaries are big, and instruction caches need to be big to get good cache hit rates. Though there is some special "MIPS16" mode in some MIPS processes which allows some instructions to be 16-bit long, decreasing code size

ARM is an instruction set which is often said to be "RISC" but it's mostly designed to be a pragmatic instruction set, borrowing only the best RISC ideas, never trying to be a "pure" RISC like MIPS or SPARC.

ARM has (and has had) quite many clever features that actually help it to achieve either better performance or lower power. Like MIPS, it cannot operate directly with values in memory, needing to load them with separate load instructions. But like x86, those memory operations have much wider variety of addressing modes, for example pre-increment and post-decrement and array indexing (A+(B<<N)). The original version of ARM also allowed almost all instructions to be predicated (executed conditionally) which allowed removal of quite a many branches. However, this feature has been removed from the latest ARMv8, but ARMv8 still has quite practical conditional select and compare operations. ARM also has load instructions which can load multiple values at once into multiple registers.

Because ARM does much more in one instruction than traditional RISC instruction set, like x86, also the ARM code is quite dense, giving good instruction cache hit rates and saving power when fetching the instructions.

Uses?

MIPS processors are used in embedded systems such as residential gateways and routers. Originally, MIPS was designed for general-purpose computing. During the 1980s and 1990s, MIPS processors for personal, workstation, and server computers were used by many companies such as Digital Equipment Corporation, MIPS Computer Systems, NEC, Pyramid Technology, SiCortex, Siemens Nixdorf, Silicon Graphics, and Tandem Computers.

Historically, video game consoles such as the Nintendo 64, Sony PlayStation, PlayStation 2, and PlayStation Portable used MIPS processors. MIPS processors also used to be popular in supercomputers during the 1990s, but all such systems have dropped off the TOP500 list.

These uses were complemented by embedded applications at first, but during the 1990s, MIPS became a major presence in the embedded processor market, and by the 2000s, most MIPS processors were for these applications.

In the mid- to late-1990s, it was estimated that one in three RISC microprocessors produced was a MIPS processor. By the late 2010s, MIPS machines were still commonly used in embedded markets, including automotive, wireless router, LTE modems (mainly via MediaTek), and microcontrollers (for example the PIC32M). They have mostly faded out of the personal, server, and application space.