Overview of the MIPS Architecture

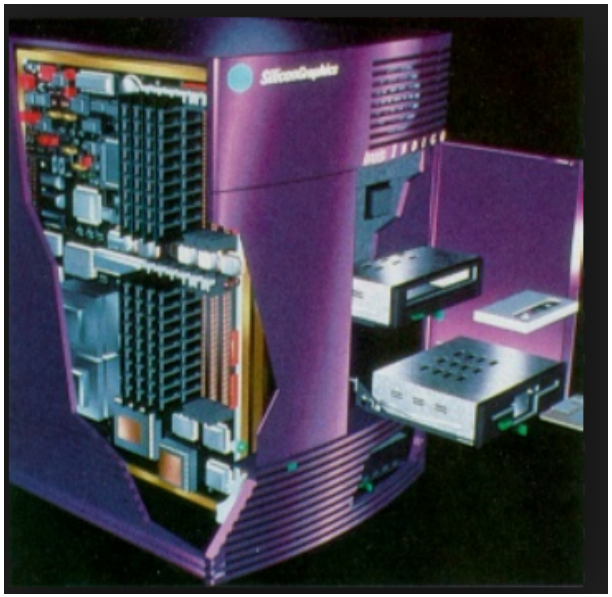
This course is based on the MIPS R3000 processor as an example of a RISC System

Sample applications of MIPS processors

MIPS processors are commonly found in:

- Set-top boxes
- Game consoles
- VoIP SoCs
- Digital TVs and DVD recorders/players
- Workstations

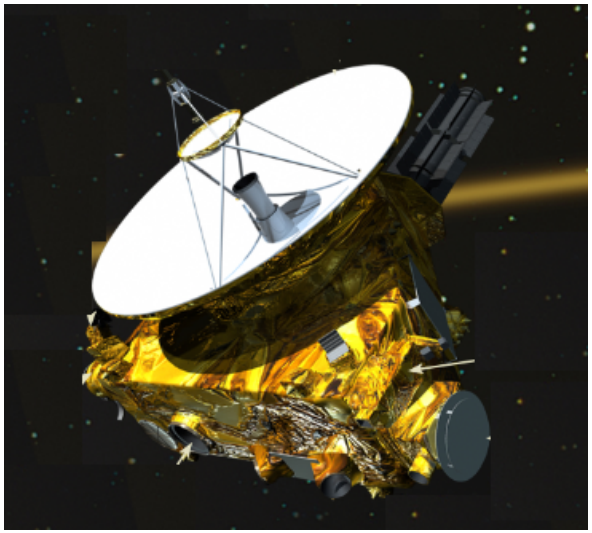
Sample applications of MIPS processors

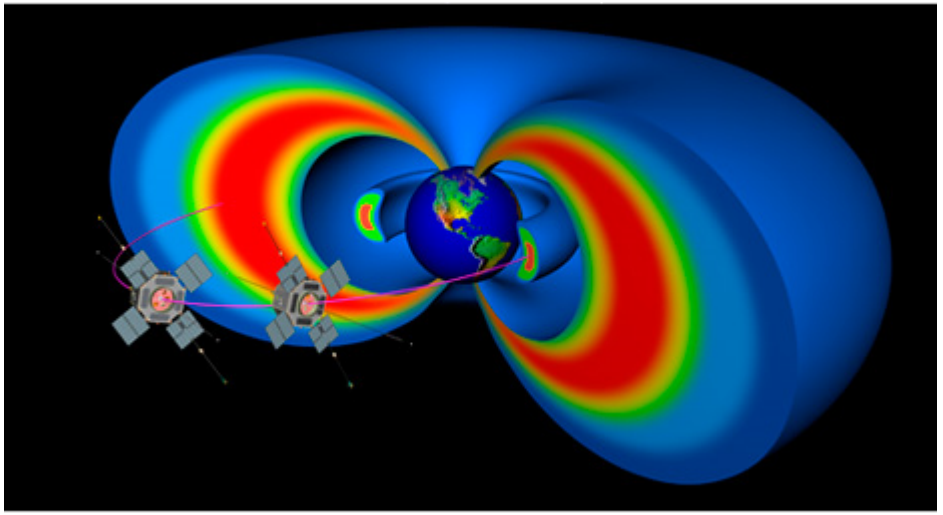


Sillicon Graphics (SGI) Workstations were based on MIPS R4000 processors, as were Nintendo 64 game consoles.

The R4000 was the first commercially available 64-bit microprocessor.

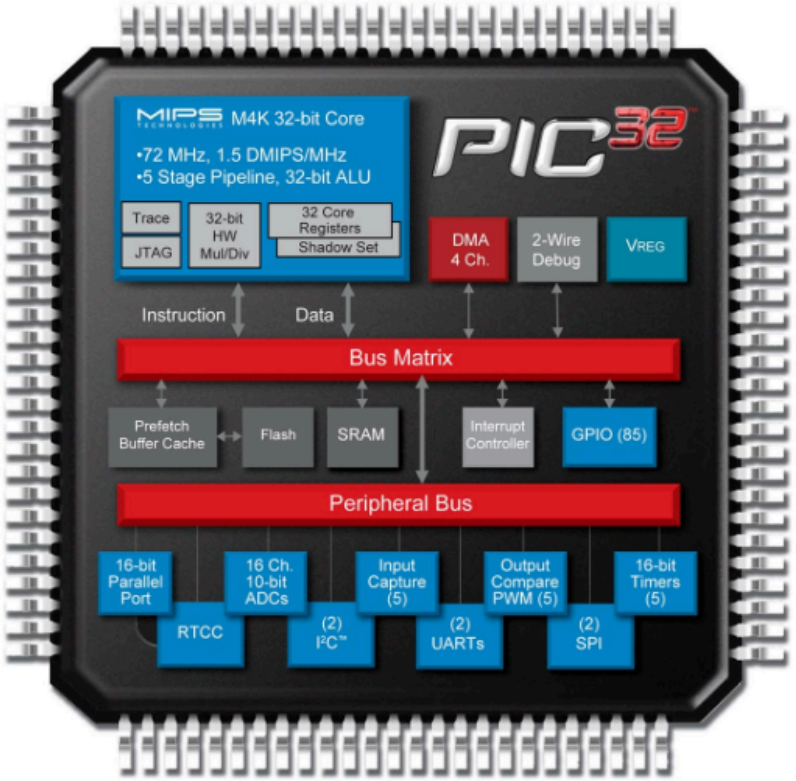
Sample applications of MIPS processors





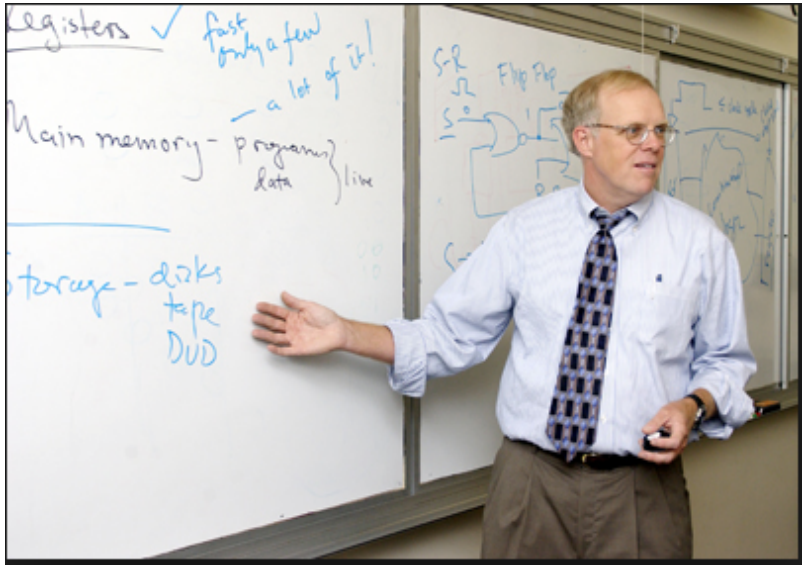
MIPS processors control the subsystems on the New Horizons Spacecraft to Pluto and were used to implement Software Defined Radios on the twin Van Allen Probes.

MIPS targets high-performance embedded designs



Used in digital consumer products and has a growing presence in mobile devices

MIPS has become a standard and performance leader in the embedded industry.



The design of the original MIPS processor was developed by Stanford University professor of engineering John Hennessy

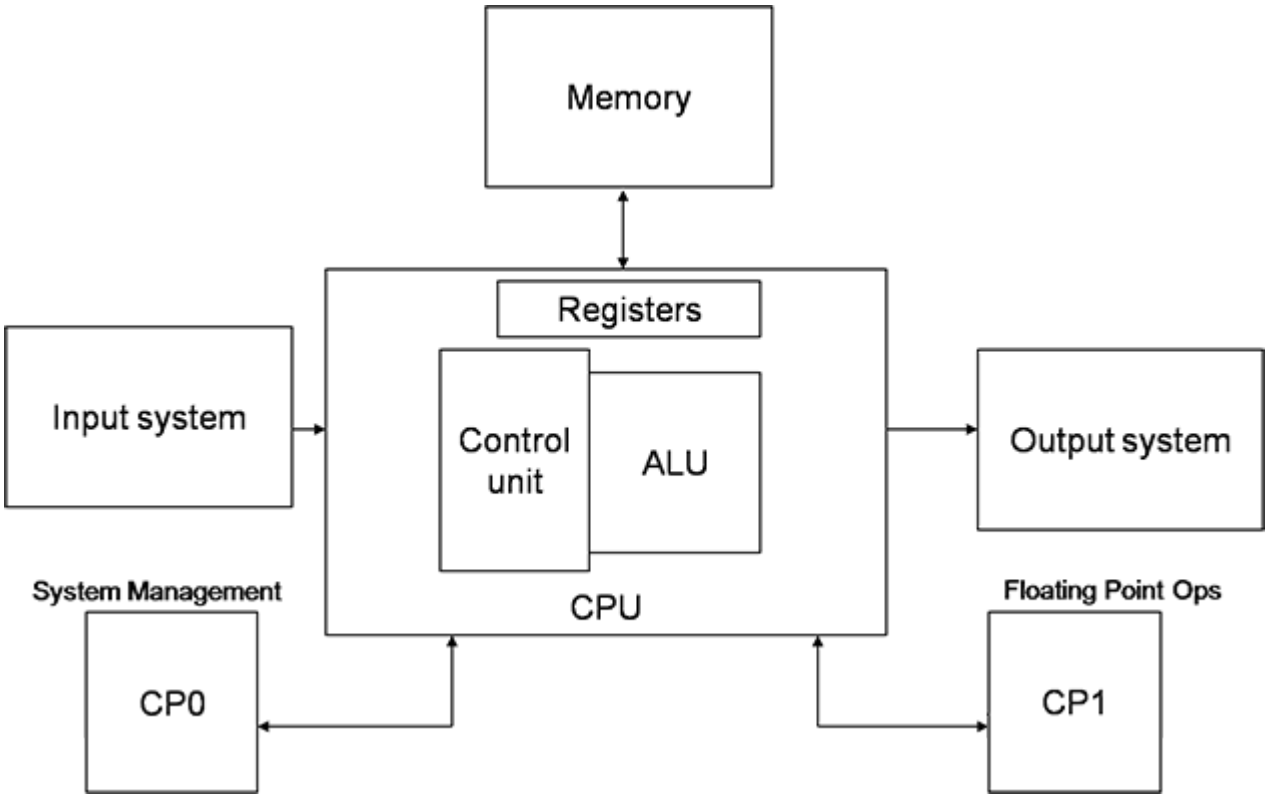
The MIPS R3000 processor is a 32-bit machine with the following RISC-like features:

- Fixed size machine instructions (32 bits)
- Only load and store instructions can reference memory (load/store architecture)
- Uses 32-bit addresses
- Pipelined functional units

MIPS RISC-like features:

- only three formats for machine instructions, all of which are 32 bits
- only four addressing modes
- 32-bit registers ($0, $1, ... , $31, Hi and Lo)

MIPS Organization



MIPS Registers

- CPU contains 32 registers, each 32 bits wide
- Registers may be used for general purposes
- MIPS calling convention should be adhered to if the user program is to be compatible with library routines and system software.

- Either register names or numbers may be used within assembly language statements
- Format: $n, where n is the register number (0 – 31)
- Register numbers are encoded in 5-bit fields within machine instructions
- Register names serve as more mnemonic aliases for the register numbers within assembly language statements

MIPS Registers

- Register $0 ($zero) is hardwired to zero
- (reads as 0 and cannot be overwritten)

- Serves a convenient source or a zero constant

Example use: add $3,$2,$0 has same effect as
move $3,$2 to copy $2 into $3

- Register $1 ($at) is used by the assembler to implement pseudo-instructions (i.e. synthetic instructions)

MIPS Registers

- Temporary registers are not preserved across function calls:

  $t0 - $t7 (register numbers 8 – 15)

  $t8 - $t9 (register numbers 24 – 25)

- Saved registers must be saved and restored if used within a function:

  $s0 - $s7 (register numbers 16 – 23)

Role of Registers in Procedure Calling

Steps required:

1. Place parameters in registers
2. Transfer control to procedure
3. Acquire storage for procedure
4. Perform procedure's function
5. Place result in register for caller
6. Return to place of call

Role of Registers in Procedure Calling

$a0 – $a3: arguments (reg's 4 – 7)

$v0, $v1: result values (reg's 2 and 3)

$t0 – $t9: temporaries (can be overwritten by callee)

$s0 – $s7: Must be saved/restored by callee

$gp: global pointer for static data (reg 28)

$sp: stack pointer (reg 29)

$fp: frame pointer (reg 30)

$ra: return address (reg 31)