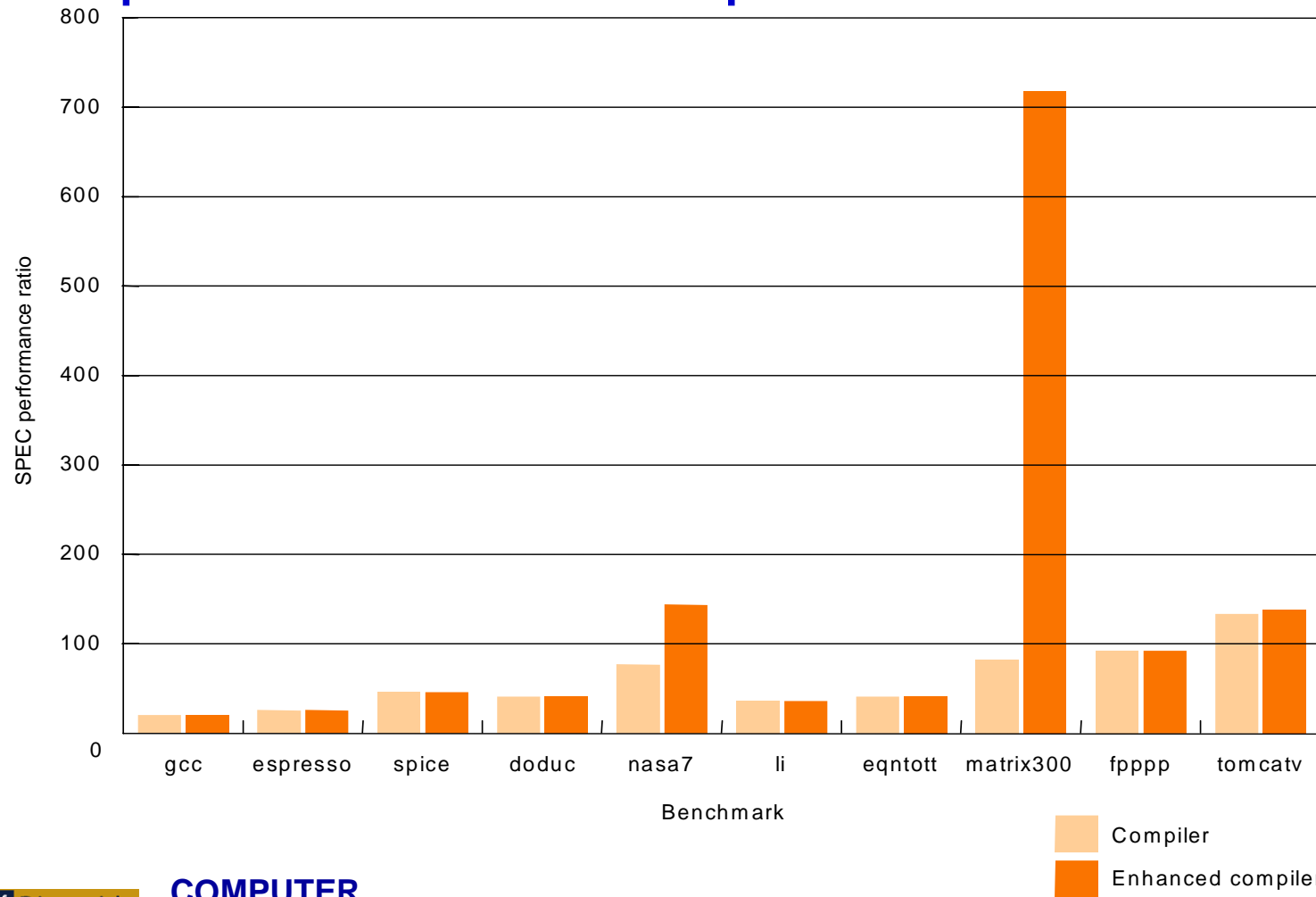


Benchmarks

- ❑ **Performance best determined by running a real application**
 - Use programs typical of expected workload
 - Or, typical of expected class of applications
e.g., compilers/editors, scientific applications, graphics, etc.
- ❑ **Small benchmarks**
 - nice for architects and designers
 - easy to standardize
 - can be abused
- ❑ **SPEC (System Performance Evaluation Cooperative)**
 - companies have agreed on a set of real program and inputs
 - valuable indicator of performance (and compiler technology)
 - can still be abused

SPEC '89

□ Compiler “enhancements” and performance



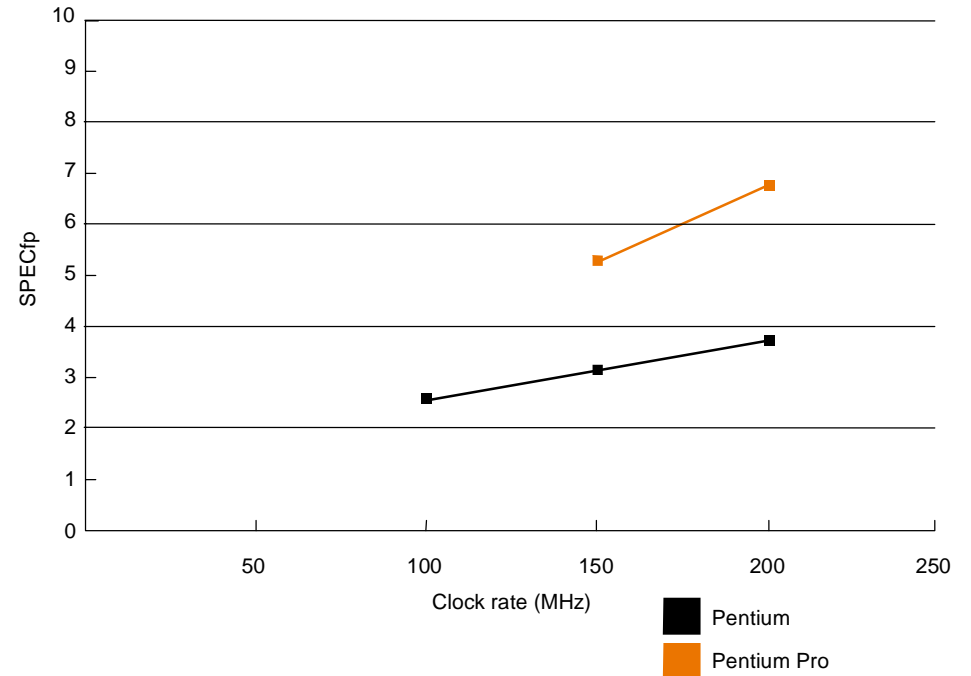
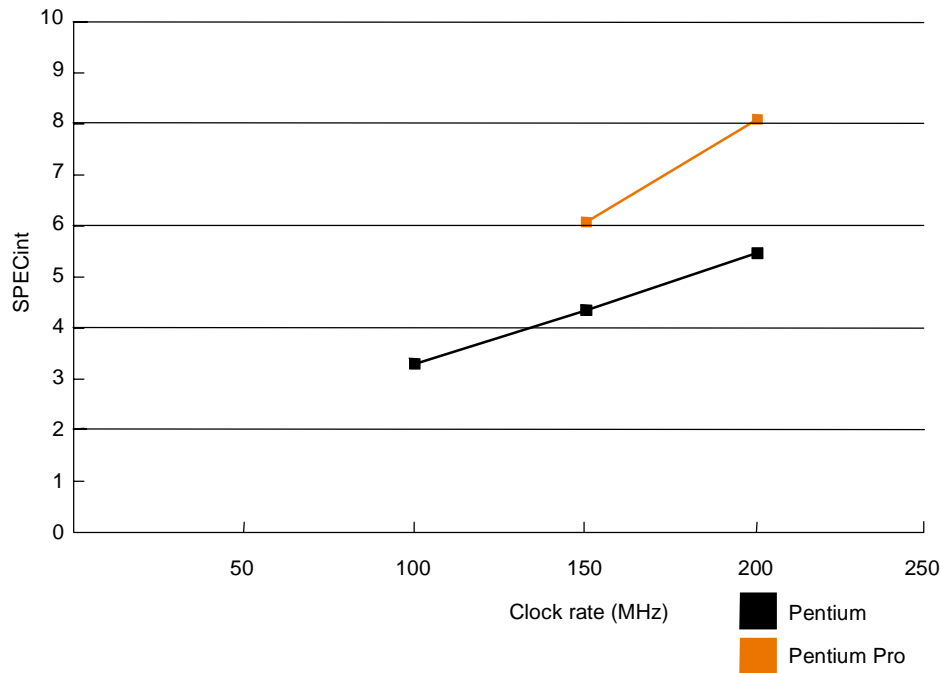
SPEC '95

Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
jpeg	Graphic compression and decompression
perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
vortex	A database program
tomcatv	A mesh generation program
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Navier Stokes equations
mgrid	Multigrid solver in 3-D potential field
applu	Parabolic/elliptic partial differential equations
trub3d	Simulates isotropic, homogeneous turbulence in a cube
apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant
fpppp	Quantum chemistry
wave5	Plasma physics; electromagnetic particle simulation

SPEC '95

Does doubling the clock rate double the performance?

Can a machine with a slower clock rate have better performance?



Amdahl's Law

Execution Time After Improvement = Execution Time Unaffected + (Execution Time Affected / Amount of Improvement)



Execution time w/o E (Before)

$$\text{Speedup (E)} = \frac{\text{Execution time w/o E (Before)}}{\text{Execution time w E (After)}}$$

❑ **Example:**

"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

How about making it 5 times faster?

❑ *Principle: Make the common case fast*

Example

- Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

$$10/6$$

- We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

$$100-x+x/5 = 100/3, \quad x=83.3$$

Remember

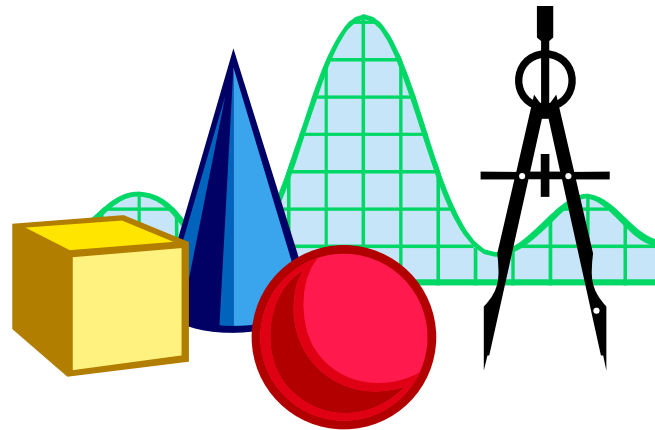
- ❑ **Performance is specific to a particular program/s**
 - **Total execution time is a consistent summary of performance**

- ❑ **For a given architecture performance increases come from:**
 - **increases in clock rate (without adverse CPI affects)**
 - **improvements in processor organization that lower CPI**
 - **compiler enhancements that lower CPI and/or instruction count**

- ❑ **Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance**

Chapter 3 – Arithmetic for Computers

Floating-point arithmetic



- ❑ **Floating-point programming in MIPS.**
 - Floating point greatly simplifies working with large (e.g., 2^{70}), small (e.g., 2^{-17}) numbers and fractional numbers (e.g. 3.14).
 - Early machines did it in software with “scaling factors”
- ❑ **We’ll focus on the IEEE 754 standard for floating-point arithmetic.**
 - How FP numbers are represented
 - Limitations of FP numbers
 - FP addition and multiplication

Floating-point representation

- IEEE numbers are stored using a kind of scientific notation.

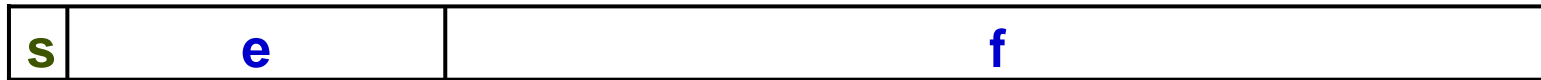
$$\pm \text{mantissa} * 2^{\text{exponent}}$$

- We can represent floating-point numbers with three binary fields: a sign bit **s**, an exponent field **e**, and a fraction field **f**.



- The IEEE 754 standard defines several different precisions.
 - **Single precision numbers** include an 8-bit exponent field and a 23-bit fraction, for a total of 32 bits.
 - **Double precision numbers** have an 11-bit exponent field and a 52-bit fraction, for a total of 64 bits.

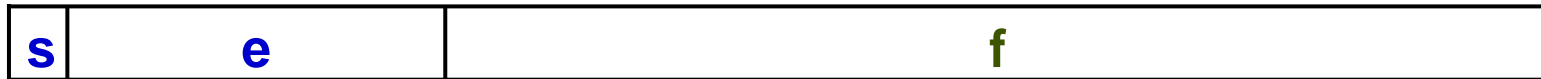
Sign



- ❑ The **sign bit** is 0 for positive numbers and 1 for negative numbers.
- ❑ But unlike integers, IEEE values are stored in **signed magnitude** format.



Mantissa

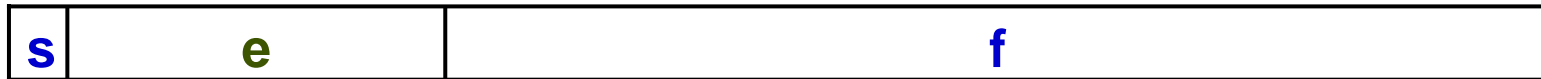


- ❑ The field **f** contains a binary fraction.
- ❑ The actual mantissa of the floating-point value is $(1 + f)$.
 - In other words, there is an implicit 1 to the left of the binary point.
 - For example, if **f** is **01101...**, the mantissa would be **1.01101...**
- ❑ There are many ways to write a number in scientific notation, but there is always a *unique normalized* representation, with exactly one non-zero digit to the left of the point.

$$0.232 * 10^3 = 23.2 * 10^1 = 2.32 * 10^2 = \dots$$

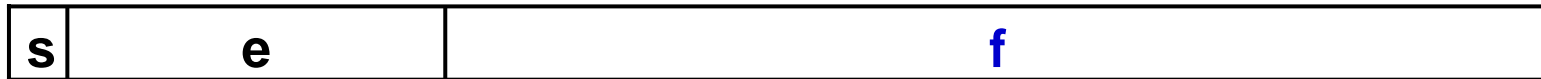
- ❑ A side effect is that we get a little more precision: there are 24 bits in the mantissa, but we only need to store 23 of them.

Exponent



- The **e** field represents the exponent as a **biased number**.
 - It contains the actual exponent *plus* 127 for single precision, or the actual exponent *plus* 1023 in double precision.
 - This converts all single-precision exponents from -127 to +127 into unsigned numbers from 0 to 254, and all double-precision exponents from -1023 to +1023 into unsigned numbers from 0 to 2046.
- Two examples with single-precision numbers are shown below.
 - If the exponent is 4, the **e** field will be $4 + 127 = 131$ (10000011_2).
 - If **e** contains 01011101 (93_{10}), the actual exponent is $93 - 127 = -34$.
- Storing a biased exponent means we can compare IEEE values as if they were signed integers.

Converting an IEEE 754 number to decimal



- The decimal value of an IEEE number is given by the formula:

$$(1 - 2s) * (1 + f) * 2^{e-\text{bias}}$$

- Here, the s, f and e fields are assumed to be in decimal.
 - (1 - 2s) is 1 or -1, depending on whether the sign bit is 0 or 1.
 - We add an implicit 1 to the fraction field f, as mentioned earlier.
 - Again, the bias is either 127 or 1023, for single or double precision.

Example IEEE-decimal conversion

- ❑ Let's find the decimal value of the following IEEE number.

1 01111100 110000000000000000000000

- ❑ First convert each individual field to decimal.
 - The sign bit s is 1.
 - The e field contains $01111100 = 124_{10}$.
 - The mantissa is $0.11000... = 0.75_{10}$.
- ❑ Then just plug these decimal values of s , e and f into our formula.

$$(1 - 2s) * (1 + f) * 2^{e-bias}$$

- ❑ This gives us $(1 - 2) * (1 + 0.75) * 2^{124-127} = (-1.75 * 2^{-3}) = -0.21875.$

Converting a decimal number to IEEE 754

□ What is the single-precision representation of 347.625?

1. First convert the number to binary: $347.625 = 101011011.101_2$.
2. Normalize the number by shifting the binary point until there is a single 1 to the left:

$$101011011.101 \times 2^0 = 1.01011011101 \times 2^8$$

3. The bits to the right of the binary point comprise the fractional field f .
4. The number of times you shifted gives the exponent. The field e should contain: **exponent + 127**.
5. Sign bit: 0 if positive, 1 if negative.