Floating Point

- Real numbers

$$3.14159 \ (\pi)$$
$$0.000000001 (1.0 \times 10^{-9})$$
$$2.71828 \ (e)$$

Floating Point

- Real numbers

$$3.14159 \ (\pi)$$
$$0.000000000001(1.0 \times 10^{-9})$$
$$2.71828 \ (e)$$

- *Floating numbers*: position of binary point is not fixed. Just like **float in C**.
- vs. "fixed-point" systems

Floating Point

- Real numbers

$$\begin{aligned}&3.14159 \ \ (\pi)\\&0.000000000001(1.0\times 10^{-9})\\&2.71828 \ \ (e)\end{aligned}$$

- *Floating numbers*: position of binary point is not fixed. Just like **float in C**.
- vs. "fixed-point" systems
- Scientific notation

Floating Point

- Real numbers

$$3.14159 \ (\pi)$$
$$0.000000001 (1.0 \times 10^{-9})$$
$$2.71828 \ (e)$$

- *Floating numbers*: position of binary point is not fixed. Just like **float in C**.
- vs. "fixed-point" systems
- Scientific notation
  - Normalized $\Rightarrow$ no leading 0

Floating Point

- Real numbers

$$\mathtt{3.14159}\ (\pi)$$
$$\mathtt{0.000000001}(1.0 \times 10^{-9})$$
$$\mathtt{2.71828}\ (e)$$

- *Floating numbers*: position of binary point is not fixed. Just like **float in C**.
- vs. "fixed-point" systems
- Scientific notation
  - Normalized $\Rightarrow$ no leading 0
- Exponent $\Rightarrow$ no. of positions to move the point in the fraction

Advantages of Normalized Scientific Notation

- Simplifies exchange of floating point data
- Simplifies arithmetic
- Increases accuracy: unnecessary leading 0's are replaced by real numbers on the right

Binary Floating Numbers

- Binary point (analogous to *decimal* point)

$$1.101_{two} \times 2^{-4}$$

Binary Floating Numbers

- Binary point (analogous to *decimal* point)

$$1.101_{two} \times 2^{-4}$$

- In general

$$1.xxxxxxx_{two} \times 2^{yyyy}$$

Binary Floating Numbers

- Binary point (analogous to *decimal* point)

$$1.101_{two} \times 2^{-4}$$

- In general

$$1.xxxxxxx_{two} \times 2^{yyyy}$$

  - Why 1 in fraction?
  - (Will use exponent in decimal for simplicity)

Binary Floating Numbers

- In design: compromise between sizes of *fraction* and *exponent*
    - between *precision* and *range*
    - since fixed word size

Binary Floating Numbers

- In design: compromise between sizes of *fraction* and *exponent*
  - between *precision* and *range*
  - since fixed word size
- Represent in (floating) binary word as:

$$(-1)^{\mathtt{S}} \times \mathtt{F} \times 2^{\mathtt{E}}$$

- S (sign bit): 1 bit (31st bit)
- E (exponent): 8 bits (bits 23 to 30)
- F (significand, fraction): 23 bits (bits 0 to 22) **literal storage**

Binary Floating Numbers

- In design: compromise between sizes of *fraction* and *exponent*
    - between *precision* and *range*
    - since fixed word size
- Represent in (floating) binary word as:

$$(-1)^{\mathtt{S}} \times \mathtt{F} \times 2^{\mathtt{E}}$$

- S (sign bit): 1 bit (31st bit)
- E (exponent): 8 bits (bits 23 to 30)
- F (significand, fraction): 23 bits (bits 0 to 22) **literal storage**
- Not just MIPS formats: *IEEE 754 floating-point standard*

Overflow & Underflow

- Range: $\mathtt{2.0}_{\mathtt{ten}} \times \mathtt{10}^{-38}$ to $\mathtt{2.0}_{\mathtt{ten}} \times \mathtt{10}^{38}$

Overflow & Underflow

- Range: $\mathtt{2.0_{ten}} \times \mathtt{10}^{-38}$ to $\mathtt{2.0_{ten}} \times \mathtt{10}^{38}$
- **Overflow**: Too large to represent
  - exponent too large to fit in 8 bits

Overflow & Underflow

- Range: $\texttt{2.0}_{\texttt{ten}} \times \texttt{10}^{-38}$ to $\texttt{2.0}_{\texttt{ten}} \times \texttt{10}^{38}$
- **Overflow**: Too large to represent
  - exponent too large to fit in 8 bits
- **Underflow**: Too accurate to represent
  - Negative exponent too large to fit

`double` format

- double-precision floating-point

`double` format

- double-precision floating-point
  - vs. single-precision

`double` format

- double-precision floating-point
  - vs. single-precision
- Uses two MIPS words

`double` format

- double-precision floating-point
  - vs. single-precision
- Uses two MIPS words
  - S: 31st bit of 1st register
  - E: bits 30 to 20 of 1st register
  - F: rest 20 bits of 1st register + 32 bits of 2nd

double format

- double-precision floating-point
  - vs. single-precision
- Uses two MIPS words
  - S: 31st bit of 1st register
  - E: bits 30 to 20 of 1st register
  - F: rest 20 bits of 1st register + 32 bits of 2nd
- Increased range:
  $2.0_{ten} \times 10^{-308}$ to $2.0_{ten} \times 10^{308}$

Another Optimization

- Normalized ⇒ Make leading 1-bit implicit

[1] as represented in the word

Another Optimization

- Normalized ⇒ Make leading 1-bit implicit
  - $\therefore$ 24 bits for significand
  - 53 bits for double-precision

[1] as represented in the word

Another Optimization

- Normalized $\Rightarrow$ Make leading 1-bit implicit
  - $\therefore$ 24 bits for significand
  - 53 bits for double-precision
- Also use *biased notation* for exponent instead of *two's complement*

[1] as represented in the word

Another Optimization

- Normalized $\Rightarrow$ Make leading 1-bit implicit
  - $\therefore$ 24 bits for significand
  - 53 bits for double-precision
- Also use *biased notation* for exponent instead of *two's complement*
- Why?
  - $\therefore$, Exponent[1] = Actual + 127
  - Bias 1023 for double precision

[1] as represented in the word

Another Optimization

- Normalized ⇒ Make leading 1-bit implicit
  - ∴ 24 bits for significand
  - 53 bits for double-precision
- Also use *biased notation* for exponent instead of *two's complement*
- Why?
  - ∴, Exponent[1] = Actual + 127
  - Bias 1023 for double precision
  - `0000 0000` is for 0
  - 1111 1111 is for infinity (could be negative or positive)

[1] as represented in the word

Another Optimization

- Normalized ⇒ Make leading 1-bit implicit
  - ∴ 24 bits for significand
  - 53 bits for double-precision
- Also use *biased notation* for exponent instead of *two's complement*
- Why?
  - ∴, Exponent[1] = Actual + 127
  - Bias 1023 for double precision
  - `0000 0000` is for 0
  - 1111 1111 is for infinity (could be negative or positive)

[1] as represented in the word

IEEE 754 Representation

- Final representation:

$$(-1)^{\mathtt{S}} \times (1 + \mathtt{F}) \times 2^{(\mathtt{E}-127)}$$

MIPS Instruction support for floating point numbers

- To load into memory (`.data` section)
  - `.float number`$_1$
  - `.double number`$_2$
- Floating-point registers:
  - `$f0, $f1, $f2, ...`
  - Use couples for double
- To load & store from memory
  - `lwc1 $f0, 0($t1)` or `lwc1 $f0, num_var`
  - `swc1$f2,0($t2)`
- For arithmetic (single precision)
  - `add.s, sub.s, mul.s, div.s`
  - `add.d, sub.d, mul.d, div.d`